

**NASA CONTRACTOR
REPORT**

NASA CR-1284



NASA CR-1284

0060506



TECH LIBRARY KAFB, NM

**LOAN COPY: RETURN TO
AFWL (WLIL-2)
KIRTLAND AFB, N MEX**

HAND-PRINTED INPUT FOR ON-LINE SYSTEMS

by M. I. Bernstein and H. L. Howell

Prepared by

SYSTEM DEVELOPMENT CORPORATION

Santa Monica, Calif.

for Electronics Research Center

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION • WASHINGTON, D. C. • MARCH 1969



HAND-PRINTED INPUT FOR ON-LINE SYSTEMS

By M. I. Bernstein and H. L. Howell

Distribution of this report is provided in the interest of information exchange. Responsibility for the contents resides in the author or organization that prepared it.

Prepared under Contract No. NAS 12-526 by
SYSTEM DEVELOPMENT CORPORATION
Santa Monica, Calif.

for Electronics Research Center

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

ABSTRACT

This document describes a program for recognizing hand-printed information in real time, which provides on-line computer users with a means of inputting two-dimensional information as simply as writing with pen and paper. The program operates under the Time-Sharing System on the Q-32 computer at SDC, and uses a RAND Tablet for input and a CRT display (rear-projected on the tablet) for output. Each user of the program builds a unique character dictionary, based on samples of his own input characters. For each user, the program currently recognizes about 100 different characters, which are chosen from a larger alphabet by the individual user. This document describes how the recognition program interfaces with the Time-Sharing System; what functions the program performs in recognizing hand-drawn input; and how the character dictionary is constructed and tested. The report concludes by suggesting that the character recognizer will realize its greatest potential by being applied to problems that require free-form (rather than linear keyboard) input.

FOREWORD

Distribution of this report is provided in the interest of information exchange and should not be construed as endorsement by NASA of the material presented. Responsibility for the contents resides with the organization that prepared it.

The work reported herein was monitored by:

Mr. David Kipping
Technical Monitor
NAS 12-526
Electronics Research Center
575 Technology Square
Cambridge, Massachusetts 02139

TABLE OF CONTENTS

<u>SECTION</u>	<u>PAGE</u>
1. INTRODUCTION	1
2. DATA FLOW AND CONTROL UNDER TSS	1
3. THE RECOGNIZER PROGRAM	3
3.1 Data Input Requirements	3
3.2 Primary (Path) Feature Extraction.	5
3.3 Shape Feature Extraction	7
3.4 Multi-Stroke Characters and Inter-Stroke Relationships.	8
4. THE DICTIONARY	10
4.1 Dictionary Construction.	10
4.2 User Dictionary Manipulation	12
4.3 Dictionary Testing	15
5. CONCLUSIONS AND RECOMMENDATIONS.	15
6. BIBLIOGRAPHY	17

TABLE OF CONTENTS - APPENDIX

1. INTRODUCTION	21
2. ANALYSIS	21
3. NOTATION	22
4. TABLES	24
5. PROGRAM DESCRIPTION.	25
5.1 Data Flow	25
5.2 Response Time.	30
6. DESCRIPTION OF PROGRAM SEGMENTS.	31
6.1 GRID Subroutine.	31
6.2 SAMPLE Subroutine	32
6.3 TEST Subroutine	32
6.4 ANALYZER Subroutine.	40
6.5 STRØKE Subroutine.	42
6.6 BOTH Subroutine.	44

TABLE OF CONTENTS - APPENDIX (Continued)

<u>SECTION</u>	<u>PAGE</u>
6.7	INFLEX Subroutines 44
6.8	XOVER Subroutine 48
6.9	MINPTS Subroutine. 48
6.10	QUAD, APUT and DIST. 51
6.11	DEFINE Subroutine. 51
6.12	SEARCHD Subroutine 51
6.13	SEARCHF Subroutine 56
6.14	SEARCHS Subroutine 56
6.15	HED8 and DIRQ Subroutines. 56
6.16	PURGE, MERGE and OPTIMIZE SUBROUTINES. 56
6.16.1	PURGE Subroutine 61
6.16.2	COMPACT Subroutine 61
6.16.3	MERGE Subroutine 64
6.16.4	Optimization 64
6.16.5	Optimize Subroutine. 66
6.16.5.1	DTREE Subroutine 66
6.16.5.2	ADDGROUP Subroutine. 70
6.16.5.3	GTREE Subroutine 72
6.16.5.4	GCHECK Subroutine. 72
6.16.5.5	RELINK Subroutine. 75
ADDENDUM A:	GLOSSARY OF MNEMONICS AND ABBREVIATIONS. 78
ADDENDUM B:	EXAMPLE OF OPTIMIZE. 85
ADDENDUM C:	NEW TECHNOLOGY 108

LIST OF FIGURES

<u>FIGURE</u>	
1.	Examples of Stages of Input Processing 4
2.	Sample Data Points for Hand-Drawn "8" 9
3.	Sample Data Points for Hand-Drawn "3" 9
4.	Value Assignments for Eight Quantized Directions 9
5.	Position Relationships Between Strokes of a Multi-Stroke Character 11
6.	Dictionary Construction. 13

LIST OF FIGURES - APPENDIX

1.	Character Recognizer Routine Relationships 26
2.	Storage Map for Display Buffer (DB) and Input Memory Buffer (IMB) 33
3.	GRID Subroutine 34
4.	Display for SAMPLE Program 35
5.	Character Subset Keyboards 36

LIST OF FIGURES - APPENDIX (Continued)

<u>FIGURE</u>		<u>PAGE</u>
6.	SAMPLE Subroutine	37
7.	Display Buffer Allocation for SAMPLE	38
8.	Display Buffer Allocation for TEST.	38
9.	TEST Subroutine	39
10.	ANALYZER Subroutine	41
11.	STROKE Subroutine	43
12.	BOTH Subroutine	45
13.	Segment End-Point Areas	46
14.	Segment Feature Areas	46
15.	INFLEX Subroutine	47
16.	XOVER Subroutine.	49
17.	MINPTS Subroutine	50
18.	QUAD ($\Delta x, \Delta y$), APUT (x), and DIST (i,j) Subroutines.	52
19.	DEFINE Subroutine	53
20.	SEARCHD Subroutine.	54
21.	Examples of Overlap Computation	55
22.	SEARCHF Subroutine.	58
23.	Character Definition Dictionary	57
24.	SEARCHS Subroutine.	58
25.	HED8 (Δxy) Subroutine Flow Chart and Table.	59
26.	DIRQ (Δxy) Subroutine Flow Chart and Table.	60
27.	PURGE Subroutine.	62
28.	COMPACT Subroutine.	63
29.	MERGE Subroutine.	65
30.	OPTIMIZE Subroutine	67
31.	DTREE Subroutine.	68
32.	ADDGROUP Subroutine	71
33.	GTRIE Subroutine.	73
34.	GCHECK Subroutine	74
35.	RELINK Subroutine	76
B-1.	Sample Dictionary Before and After Optimization	86
B-2.	Step 1 of Optimization.	87
B-3.	Step 2 of Optimization.	88
B-4.	Step 3 of Optimization.	89
B-5.	Step 4 of Optimization.	90
B-6.	Step 5 of Optimization.	91
B-7.	Step 6 of Optimization.	92
B-8.	Step 7 of Optimization.	93
B-9.	Step 8 of Optimization.	94
B-10.	Step 9 of Optimization.	95
B-11.	Step 10 of Optimization	96
B-12.	Step 11 of Optimization	97
B-13.	Step 12 of Optimization	98
B-14.	Step 13 of Optimization	99
B-15.	Step 14 of Optimization	100
B-16.	Step 15 of Optimization	101
B-17.	Step 16 of Optimization	102
B-18.	Step 17 of Optimization	103
B-19.	Step 18 of Optimization	104
B-20.	Step 19 of Optimization	105
B-21.	Step 20 of Optimization	106
B-22.	Final Result of Optimization.	107

1. INTRODUCTION

For the past few years, we have been working at the task of producing a program for the on-line recognition of hand-printed characters in real time. Our main goal has been to provide the on-line computer user with a more flexible input mechanism than now exists. Among the primary aims of research was the ability to recognize at least 100 different characters (chosen from a larger alphabet) for a given individual. A description of our earlier efforts, as well as other work in this field, can be found in references 1 through 9 in the bibliography.

The method described here is universal in the sense that it applies the same general analytical technique to all inputs. It is not universal in the sense that it may not recognize inputs provided by one other than the original dictionary builder. Thus, to obtain optimum performance, each individual user is required to build a dictionary based upon his own inputs. The system was designed with this facility, and dictionary building has been made as painless as possible. Though we do not claim that this is the ultimate in on-line character recognition, we do feel that we have come close to achieving our principal objective--namely a program that will recognize 100 characters for a given individual.

The program we have developed operates under the Time-Sharing System on the AN/FSQ-32 computer at SDC. The hardware required for our recognition system, in addition to a reasonably fast digital computer, includes a RAND Tablet or its equivalent for input and a CRT display for output. We have taken advantage of the fact that the Grafacon 1010A (the commercially available version of the RAND Tablet) was ported for rear projection, and have built the display and tablet around a projection system that provides a common input-viewing surface [10]. In use, the tablet behaves much like pen and paper. This latter feature is not essential to the technique, but we feel that it aids materially in achieving the close coupling desired in interactive man-machine systems.

2. DATA FLOW AND CONTROL UNDER TSS

The hardware for the SDC Time-Sharing System (TSS) consists of two computers, a PDP-1 and the Q-32, coupled by a core storage (called Input Memory) common to both. All interactive devices, including our Graphic Tablet Display Console, are connected to the Q-32 via the PDP-1, which serves solely as an I/O processor for these devices. Inputs from the tablet are processed by the PDP-1 CPU on an interrupt basis. Output to the display is handled through an independent controller attached directly to the Input Memory. Immediate feedback is thus provided between the tablet and display so that the user may see his actions as they occur, by having the PDP-1 store the processed tablet inputs into Input Memory in the area reserved as the display's buffer.

There are three kinds of feedback provided by the tablet interface program in the PDP-1. The first is negative--no feedback at all, indicating that input will not be accepted at that time. Next, there is feedback in the form of a

single point output, updated every 30 msec., indicating that input is allowed. The point on the display will represent the present position of the stylus on the tablet as long as the tip switch in the stylus (which we call pen switch) is open. When the pen switch is closed by pressing the stylus on the tablet surface, the sampling interval to the PDP-1 is decreased to 4 msec. The PDP-1 then stores the track or path of the pen in the Input Memory display buffer after first smoothing and filtering out redundant points (these functions are explained in detail below). To the user it appears as though "ink" were flowing from the tip of the stylus.

The control for allowing input resides with the user's program (in this case the character recognition program) in the Q-32. Communication between the user's program and the PDP-1 functions is handled through TSS's Dispatcher, using reserved words in the Input Memory display buffer. From the Q-32, the user--in addition to allowing or disallowing input (by specifying a delay time between $\frac{1}{4}$ sec and 8 sec after at least one tablet input has occurred)--also informs the PDP-1 when to notify the Q-32 system that input is finished and is waiting to be processed by the user's program. As a default condition, the time delay is ignored by the PDP-1 program if the user has filled the allotted area in the Input Memory display buffer to capacity.

All interactive user programs running under TSS are scheduled on a round robin basis. The user's program in the Q-32 issues a request for tablet input through the system's Dispatcher with a Wait in order to remain synchronized with and in control of the user's actions at the console. When the PDP-1 informs the Q-32 system that tablet inputs are ready, the user program that requested the input (only one can do so, because the tablet is acquired as a private device by that program) is taken from Wait status and is scheduled. The inputs from the tablet are then directly available on a word-for-word basis from Input Memory to the user program. In addition to the actual x,y coordinates that constitute the smoothed and filtered "ink" the user sees on his display, the PDP-1 has kept count of the number of points rejected for each accepted point in the stroke or line. This data is stored along with the x, y data in the Input Memory display buffer. Because the user may draw more than one stroke or line, the beginning of each is uniquely marked.

The Q-32 processes the input from the tablet as determined by what actions were valid at the time. The input may be interpreted as a "button" push or as a stroke in a character, or they may be rejected as invalid for the existing situation. During that time, the user will get no tablet feedback, thus informing him that input will not be accepted. Upon completion of processing, the display buffer is appropriately updated, the Dispatcher is called requesting tablet input, re-instituting the user's feedback, and the Q-32 user program re-enters Wait status.

This, in essence, describes the data flow and communication capabilities available for using the interactive Graphic Tablet Display Console within SDC's

Q-32 Time-Sharing System. Two control programs, SAMPLE, used for dictionary building, and TEST, used to test the dictionary in a simulated environment, use the communication and control features of the system as described above. The basic difference between them is the interpretation of the input data; this will become clear when they are discussed below.

3. THE RECOGNIZER PROGRAM

3.1 DATA INPUT REQUIREMENTS

The processing programs involved in the character recognizer do not deal directly with raw tablet data. Rather, they expect data that has been pre-processed in a particular way to eliminate redundant points, provide as smooth a path as possible, and yet retain the appropriate level of detail to permit extraction of pertinent features.

The functions of smoothing, filtering and keeping count of the rejected points has been relegated to the PDP-1 tablet interface program for reasons of efficiency in both time and space. Performing these tasks as each point interrupts the PDP-1 does not impose an undue burden on that processor and saves a great deal of buffer space, thus extending the "ink" supply.

Smoothing is required for the very simple reason that the raw data track from the tablet contains certain irregularities due to the discrete nature of the grid, the view angle of the stylus tip (which varies as the pen is rotated in the hand while writing) and other vagaries of electronics such as poor signal-to-noise ratio on the low-order bits of the coordinates, particularly the y coordinate. Figure 1 illustrates rather clearly the value of smoothing when column "A" is compared with column "B". Columns "C" and "D" of the same figure show the reduction in the amount of data received versus that output by filtering using a filter constant of 3. In most cases, it is obvious that the smoothed, filtered data provides the more desirable inputs for processing. The numerical data representing column "D" of Figure 1 (along with the associated point counts) are what the character recognition program processes.

The smoothing algorithm is a rather simple one. An eight-point moving average of the raw data generates one smoothed data point. To start the process, the first input point is replicated eight times. An alternative choice would have been to wait until eight points had been input. The former is logically simpler, and (because the pen is moving slowly at the beginning of a stroke), no obvious bias has been noticed because of this choice. Smoothing may be turned off by setting the appropriate control word in the display buffer.

Filtering and counting the redundant points is another simple process. It is applied after the raw data has been smoothed. The filter constant can be set into one of the display buffer control words, and can have a value between 0 and 63. Zero means no filtering. (This is the way the raw data in Figure 1 was obtained.) By trial and error, we have settled on the value of 3 for the

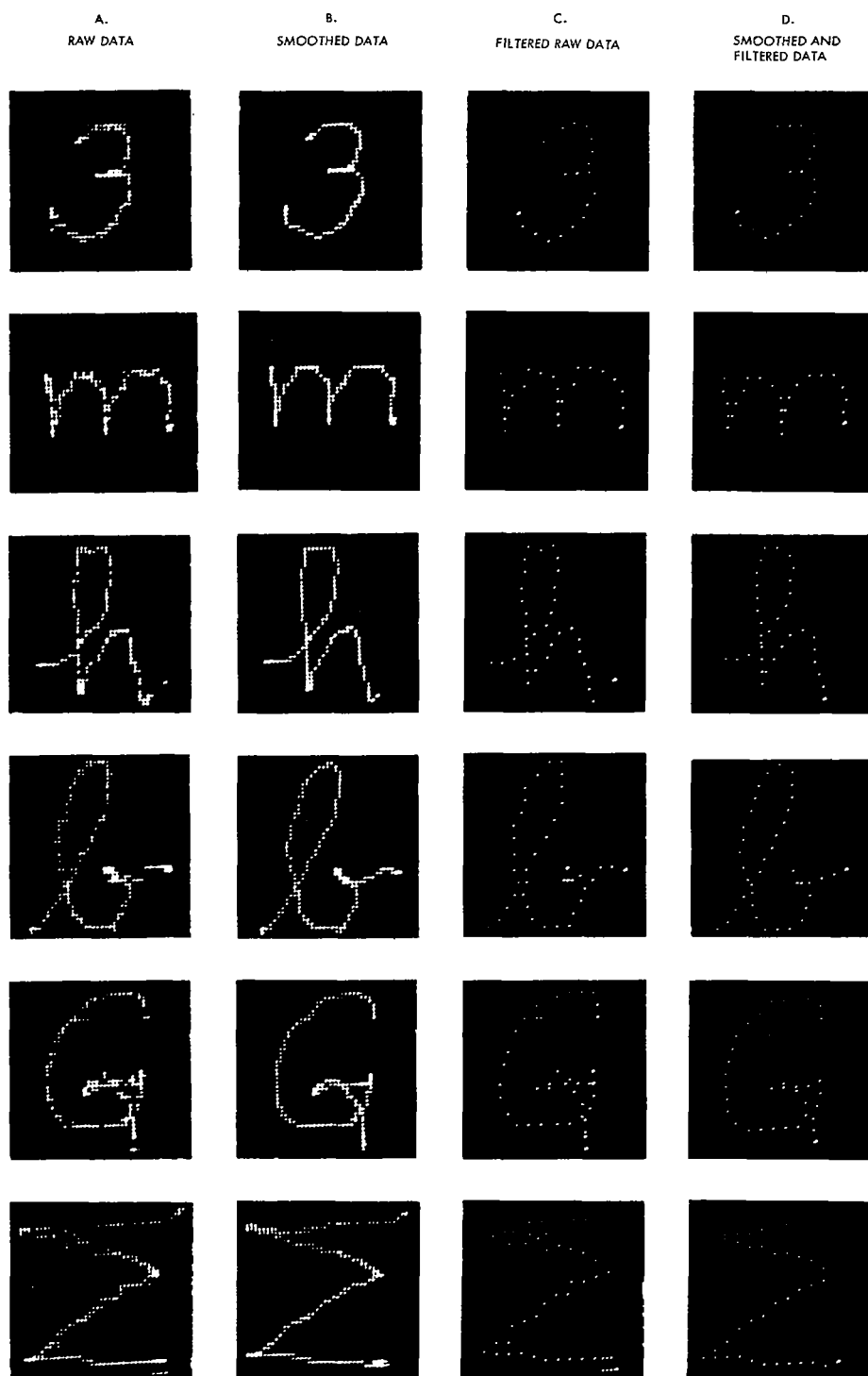


FIGURE 1. EXAMPLES OF STAGES OF INPUT PROCESSING

filtering constant as one suitable for drawing reasonably small characters while minimizing the amount of input data to be processed.

The process of filtering itself compares the absolute value of the differences between the last accepted point and the current point (output by the smoother if it is on or read directly from the tablet if it is off). If either $|\Delta x|$ or $|\Delta y|$ is greater than or equal to the filter constant, the point is accepted; otherwise it is rejected and the point count of the last accepted point is incremented. These point counts are used in (and are critical to) corner detection as a measure of the stylus velocity along the path. The filtering process is started with the first input point of the stroke. This point is also uniquely marked to identify it as the beginning of a stroke. All subsequent processing of the tablet inputs takes place in the Q-32.

3.2 PRIMARY (PATH) FEATURE EXTRACTION

The first processing of the stroke information extracts what we shall call primary or path features. At present these consist of corners, inflection points, intersections within the stroke, and a corrective procedure for removing small but bothersome "hooks" that occur at the beginning or end of the stroke. Although all of this processing could be done in one pass of the stroke data, the program would be extremely complex. Some of the processes are done in parallel; however, we shall consider them serially for clarity of exposition.

The data received by the routine is an array of ordered triples that are made up of the x-coordinate, the y-coordinate and the rejected point count. The first step is to convert the coordinate points into discrete headings (one of thirty-two)*, thus making the stroke position-independent.

* Values assigned to the 32 directions can be thought of either as simple integers having the value 0 through 31, or as signed two's complement four-bit fractions. Thus one can think of the half circle beginning at the zero direction and rotating clockwise as increasing in value by increments of $1/16$, and decreasing in the counter-clockwise direction by $-1/16$. In order to remain consistent with two's complement arithmetic, the value assigned to the direction half way around from 0 equals -1. This method permits differencing the heading using the arithmetic of most computers directly and has the advantage that no difference can exceed 180° (-1) and that the sign of the difference indicates the direction of the path's rotation or curvature. The differences themselves can be thought of either as integers or fractions, since the conversion is a simple scaling by a power of two. We shall be consistent here and treat all things concerning headings and differences as fractions.

Simultaneously, the minimum rectangle surrounding the stroke is computed. The absolute value of the difference of the headings formed by the first three and the last three points of the stroke (if the stroke area is large enough to qualify) is tested against a "hook" threshold. If the threshold is exceeded (the current value is $5/16$) the offending point is removed from the stroke, thus eliminating the "hook".

We have tried several schemes for detecting corners. All have used both local geometry and velocity. Our best estimate of velocity is the rejected point count supplied by the filtering program, which is actually the inverse of the velocity, slightly modified by the smoothing algorithm. In order to examine the local geometry of the stroke, adjacent headings are differenced, and each difference is associated with the point common to the two headings. Thus three points in a straight line would generate a zero difference for the center point, regardless of direction. In order to determine approximate angular change without regard to direction of rotation, one need only look at the absolute value of the difference. In what follows we shall use the notation Δh to denote this difference, and $|\Delta h|$ to refer to its absolute value.

The present corner detector marks a point as a corner if either its $|\Delta h|$ is greater than $11/16$ or if $|\Delta h_i + \Delta h_{i+1}|$ (that is, the sum of the current and next Δh) is greater than $13/16$ without examining the local velocity (point count). Otherwise, a point is marked as a corner if its point count is greater than or equal to 8, or the sum of its point count and its predecessor's point count is greater than or equal to 14 and this latter sum is also at least six times greater than the minimum point count two away (before or after). There are some involved complexities for those latter cases that determine which of the two candidates is actually marked as a corner, plus some point relocation that is done when the filter has "rounded" a corner, but these are not appropriate for this discussion. The parameters used in the above tests were arrived at empirically after examining a great many samples. Unfortunately, those samples did not come from a large number of people, but they have worked successfully for a varied set, including both left- and right-handed people.

It should be noted here that the marking of a point as a corner divides the stroke into separate parts from which the shape features and inflection points are extracted separately. If no corners are found in a stroke, the entire stroke is processed as a unit for extraction of shape features and inflection points.

Inflection points are determined using the same heading differences described above. This is done by summing the Δh 's for the entire stroke and noting when the absolute value of the sum exceeds $5/16$. Only after such an occurrence can an inflection point occur. This limit eliminates spurious inflections introduced by minor wiggles. After the above threshold is exceeded, the maximum and minimum values of the sum are noted if the difference between successive pairs of minima and maxima is greater than $3/8$. If so, an inflection point

has occurred either at or between the pair. The usual case is that the event occurred at the minimum or maximum, but when they are joined by a straight segment (a series of Δh 's equal to zero), the inflection point is taken midway in this segment.

We have not solved the general problem of finding path intersections efficiently at this writing. Instead, we look at the sums generated by the inflection point search and determine if any part of the path qualifies as a loop. Then, if and only if no inflection point was found in the part under question, we perform tests within the general area of the stroke to determine if closure has occurred. We hope that a general solution to the problem of detecting path intersections will be more useful in conjunction with shape feature extraction in improving discrimination among strokes that now prove somewhat troublesome to handle as special cases.

3.3 SHAPE FEATURE EXTRACTION

Over the past several years, we have tried various methods for extracting features describing the "shape" or topology (using the term loosely) of a stroke. They can be divided into three general classes: local minima and maxima, area traversal, and curvature measurements. Each has its merits and shortcomings. When the shortcomings outweighed the merits (as was the case with the local minima-maxima method), the effort was discarded from further consideration as a basic method, though we attempted to learn from such failures and retain some features of the approach that might lead to improvements of other attempts with other techniques.

The area feature extractor, which has already been documented in earlier reports, is a case in point. Though the technique as a whole was not entirely satisfactory because it was sensitive to minor variations in the input that drastically changed the generated description (such as lengthening or shortening the tail on the input character "a"), it had characteristics we wished to retain in future tries. For instance, it inherently retained the basic geometric relationships among the various parts of the stroke, and was insensitive to minor variations in the amount of curvature in the various parts of the stroke. In fact, as long as "faults" such as hooks did not generate corners or inflection points, they did not perturb the results at all.

On the other hand, the curvature feature was insensitive to lengthening or shortening "tails", but was always sensitive to minor variations in curvature if the variation occurred at the separation between two classes of features, such as a "curve" and a "cup." We had solved the problem of retaining geometric relationships between the features by adding a feature that described the relationship of the present feature to the collection of its predecessors. The details of this extractor appeared in a prior report.

The most recent feature extractor is an amalgamation of area and curvature techniques. The stroke is segmented into smaller parts at corners and (when we solve the processing problem) at intersections, but not at inflection points--as was necessary in the curvature measurement method. Each part is then "described" using the area feature method with some minor changes, and the whole is tied together. This is done by geometrically relating the rectangles used to generate the description for the parts in the same way that the features were related in the curvature extractor. The variations to this technique amount to eliminating the central diamond-shaped area if an inflection point is included in the stroke part, and marking the inflection point's occurrence instead; if the part or stroke is simply a straight line, the generated feature string contains only the end point areas. This latter case is detected when the Δh 's are summed while looking for inflection points. The same procedure permits us to detect loops that occur at the beginning and/or end of a stroke, and to treat them separately. To clarify the above, we present two examples: a hand-drawn eight, and a hand-drawn three.

Figure 2 shows the data points for the input character (the "g") and indicates that an inflection point was found (the circled point), plus the area divisions of the minimum rectangle surrounding the stroke. The feature string produced by following the path from area to area is 145I2351. The "I" indicates that an inflection point was encountered in area 5; however, since it was, all references to area 5 are deleted, thus producing the feature string 14I231. The deletion of area 5 was decided upon after many samples of strokes were analyzed for variability, and it was discovered that almost all of the differences were in the area in which the inflection was placed, plus the relative position of area 5 with respect to the others. Although not shown in this example, when the deletion of area 5 leaves two adjacent features with the same area number, one of these is also deleted. The overall effect is a decrease in undesirable discrimination, at the cost of extra processing.

The hand-drawn three shown in Figure 3 illustrates the case where the stroke has been divided into two parts by the discovery of a corner (the point surrounded by a square). The rectangle surrounding each part is computed and subdivided as shown, and the generated feature string is 4123C4123G4. The "C" indicates that a corner occurred between the two parts; the "G4" at the end shows the following geometric relationship between the two rectangles: the second part is immediately below the first. The headings or directions used for geometric relationships between surrounding rectangles is quantized to one of eight as shown in Figure 4. The overall effect of this approach is reasonably consistent shape description of parts of a stroke that is insensitive to minor variations in curvature, while retaining the overall required discrimination and descriptive content.

3.4 MULTI-STROKE CHARACTERS AND INTER-STROKE RELATIONSHIPS

Being able to extract the feature content of each stroke only solves half of the problem for multi-stroke characters. In order not to impose restrictions

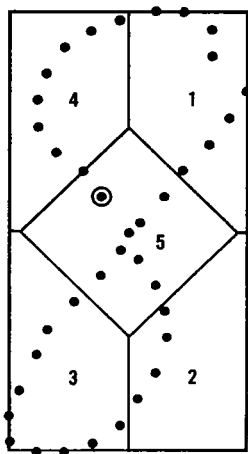


FIGURE 2: SAMPLE DATA POINTS
FOR HAND-DRAWN "8"

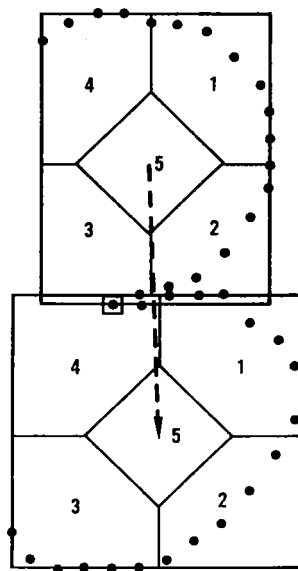


FIGURE 3: SAMPLE DATA POINTS
FOR HAND-DRAWN "3"

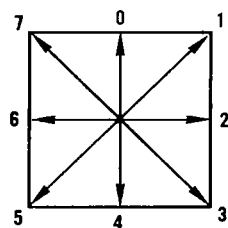


FIGURE 4: VALUE ASSIGNMENTS FOR
EIGHT QUANTIZED DIRECTIONS

or constraints upon the user concerning character separation and to make the recognizer truly position-independent, multi-stroke characters must be dealt with as a unit. Therefore, we require some additional information about the relationships of strokes that make up the character. The method used is the same as that used to describe the geometrical relationships between the parts within a stroke. Each successor stroke is related to the collection of its predecessors by computing the relationship of the appropriate minimum surrounding rectangles. A stroke is considered coincident with its predecessor(s) if the rectangle centers are within a limit of one another, as computed from the larger of the two. If the coincidence test fails, the center-to-center direction is computed; on the basis of that direction, the appropriate edges are compared for nearness (based upon the same computed limit) or overlap. If this test is successful, the strokes are considered near (indicated by "N"); otherwise they are far (indicated by "F") from one another. The position relation is composed of the result of these tests plus the center-to-center direction in the case of "near" and "far" rectangles. In Figure 5, the positional relation is indicated by the appropriate letter followed by an arrow where required.

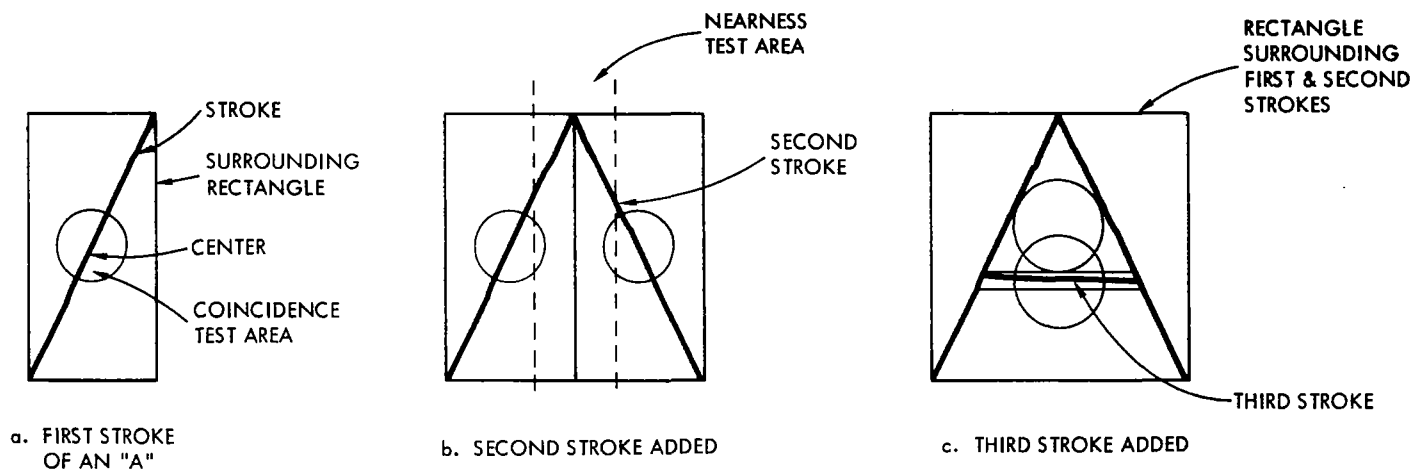
4. THE DICTIONARY

4.1 DICTIONARY CONSTRUCTION

As stated above, each user should build a dictionary based upon samples of his own input characters. This is presently done using the control program called SAMPLE. In the dictionary-building mode, only one character at a time is input, though it may be constructed of as many as 12 strokes. When the user completes his input (this is indicated by pausing a predetermined amount of time), the strokes are individually analyzed, that is, a feature string is produced for each; if the input character is composed of more than one stroke, the positional relationship between each subsequent stroke and the collection of all of its predecessors is computed.

The dictionary--as it exists--is then searched, stroke by stroke, for a matching description. If a complete match is found for the input, the associated output character replaces the user's input on the display, thus informing him that the input was recognized. If no match, or an incomplete match is made, the user is so informed. He then defines the input by appropriately indicating the output character he wanted to associate with the input; alternatively, he may erase the input and draw another character. By defining the input, the user causes the program to add the missing stroke information to the dictionary with the output character appended.

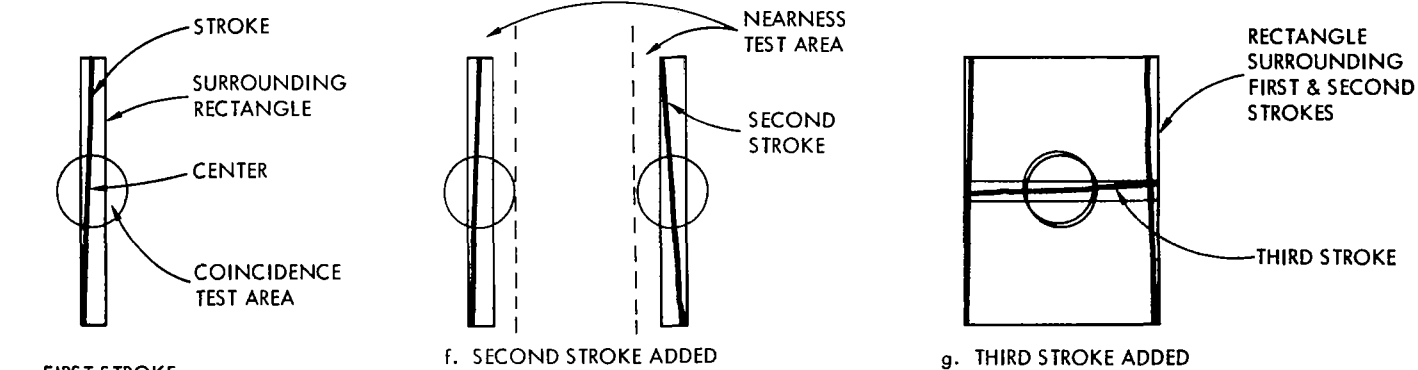
In addition to the feature information, the dictionary contains a recognition count, set to zero when a new definition is added. Each time a match is made for a definition, its recognition count is incremented by one.



$$S_2: S_1 = N \rightarrow$$

$$S_3: S_1 + S_2 = C$$

d. GENERATED POSITION RELATIONSHIPS FOR "A"



$$S_2: S_1 \approx F \rightarrow$$

$$S_3: S_1 + S_2 \approx C$$

h. GENERATED POSITION RELATIONSHIPS FOR "H"

FIGURE 5. POSITION RELATIONSHIPS BETWEEN STROKES OF A MULTI-STROKE CHARACTER

The dictionary minimizes both space and search time, while retaining the essential stroke relation information. This is done by constructing the dictionary as a tree or set of trees, in which only the legitimate successor strokes are linked to predecessors. Figure 6 illustrates the dictionary-building process. In Figure 6, the feature strings generated for the strokes are represented by drawn shapes, rather than their actual numerical representation. The dotted arrows indicate the paths that the search routine is allowed to follow. Figure 6f illustrates the parsimony that this type of dictionary allows, namely, it contains descriptions of 11 strokes and 9 complete characters without ambiguity.

As constructed by the user, the dictionary contains not only the input character descriptions and definitions, it also contains implicitly the separation information required if the user is to be permitted to input more than one character at a time. This factor is essential for two reasons: First, the user can input a natural grouping at one time; second, and even more important, he is not constrained to print his input at some predetermined bounded area.

Errors or ambiguities can and do occur, because some characters are proper subsets of others, and legitimate character pairs appear to the program as single characters. To resolve this problem, the user must revert to single character input.

4.2 USER DICTIONARY MANIPULATION

In addition to the obvious abilities to save and restore a dictionary under user control, the current program permits three other capabilities: purging, merging, and optimizing multi-stroke character definitions.

Purging permits the user to delete unwanted definitions from the dictionary in one of two ways. He may delete all definitions for a chosen output character, or he may purge all definitions whose recognition count is below a threshold of his choosing. There is no restriction on the number of times the user may purge a dictionary.

The purging is done as a two-step process that is the same for either kind of purge-threshold or character. During the first step, every entry in the dictionary is examined in its logical sequence by beginning with the first dictionary entry of the first strokes and following all links (explicit and implied) to their terminal node or leaf. Every entry that meets the purging requirement is marked as an "intermediate undefined" entry (replacing the existing character definition entry).

On the second step, a new dictionary is built from only those entries that terminate at a leaf with a legitimate output character. All chains of entries that are all "intermediate undefined" are deleted, as long as none are cross-connected to legitimate output characters. Conceptually, the process is straightforward, but because the dictionary is not constructed using the standard technique of an available space list, the actual manipulations become quite complex.

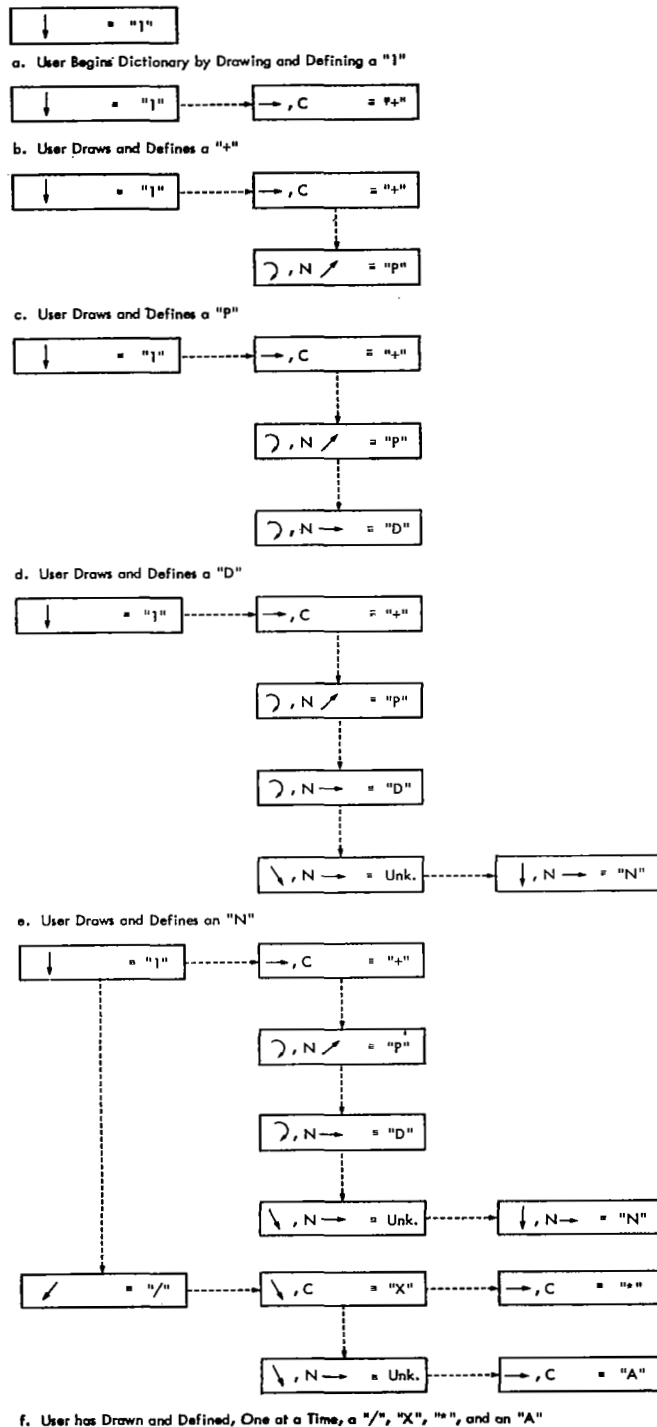


FIGURE 6. DICTIONARY CONSTRUCTION

Merging two dictionaries is a straightforward process. The dictionary that is to be added is treated as though the input characters were input one at a time, as in dictionary construction. Each complete entry (sequence of strokes) is taken one at a time from the incoming dictionary and searched for in the resident dictionary. If a complete match is found, no new information is added to the resident dictionary, but the recognition count plus one of the "new" characters is added to the resident definition. When a conflict arises--that is, when the feature strings from both dictionaries agree but the output characters differ--the user must make a choice among three alternatives: (1) He may choose to use the resident dictionary definition; (2) the definition from the incoming dictionary to replace the resident definition, or (3) neither. If he chooses neither, the resident definition is set to "intermediate undefined". If any instance of this latter occurs, the merge program calls the second step of the purge routine to clean up the dictionary.

The optimization process is a way of recovering space in the dictionary by cross-linking definitions of multi-stroke characters in a way that eliminates redundant successor stroke chains that are exactly alike. This process cannot easily be done during dictionary construction without adding a great deal more structure to the present form of the dictionary. The price for doing so would not only be additional space, but additional time during searching.

Optimization is done, in a sense, by turning the dictionary inside out. That is, the dictionary must be examined in reverse order: the last stroke in a chain and its attached definition are examined first, and all last strokes at the same level are searched for an exact match. If such a match is found, obviously one of the entries is redundant and may be removed by linking the two predecessors to the same successor. Before it is actually removed, the chain is searched backwards to determine if the redundancy continues. If such is the case, an entire chain may be removed, as long as no definitions are destroyed in the process and no ambiguities are created.

The process continues through each level of successor strokes until no further redundancies can be removed. In actual practice, the dictionary is not turned inside out, but rather it is searched in a forward direction and the entries grouped into classes which have the same number of strokes terminating in a leaf. These groups are then processed by looking first for matching definitions and then for matching feature strings. Because each definition is not necessarily a simple chain, great care is taken that no destruction or ambiguity is created; therefore, if any deletion is in the slightest way questionable, no action is taken. In practice, the optimizer has worked exceedingly well and actually created new definitions by legitimately linking together combinations that did not appear as samples during dictionary construction.

4.3 DICTIONARY TESTING

The present implementation permits the user to test his character dictionary in a simulated use mode under control of a routine called TEST. In order to better approximate a usage environment, this routine allows the simple editing functions of erasure (one or a group of characters), and single and multiple character replacement. It does not provide the ability to insert and delete, primarily because it is not line- and character-space-oriented, but rather is oriented toward unconstrained two-dimensional input. Thus the user is free to draw any character in any position at any time. Because two-dimensional notations utilize character size as well as position (for reasons of aesthetics if not meaning), the character output by the program matches the user's input in both size and aspect ratio as closely as is physically possible.

When testing a dictionary, the user is not constrained to writing one character at a time. He may input as long a character string as his "ink" supply will permit. This is the only way in which the inherent character separation mechanism of the dictionary can be tested. In this mode, the dictionary's recognition counts are incremented for each successful match as they are in the SAMPLE routine.

At present, if recognition is not acceptable, we have no way of using the test inputs for improving the dictionary. Though the recognition level may appear acceptable in the dictionary-building mode, it appears that one's actions when printing a single character differ from when he is printing a string of characters. We are planning on changing the dictionary construction program to accept a string of characters as well as single characters, and--in addition--to permit retrieval of input from the testing routine for dictionary additions. This should speed dictionary construction as well as improve its content.

5. CONCLUSIONS AND RECOMMENDATIONS

Given the appropriate input hardware, computer, and system interfaces, on-line character recognition in real time is feasible. That it can be made operationally successful when used by randomly chosen individuals has yet to be proven. We have constructed dictionaries of a large number of characters (approximately 100) for individuals actively engaged on this project, and have had acceptably high recognition rates. On the other hand, we have not specifically tested the recognizer to ascertain its maximum level of attainment, nor have we tried to have someone unfamiliar with the effort build and test a dictionary under test conditions. Demonstrations of the system for a number of visitors have indicated that some learning period is required before an individual becomes thoroughly comfortable with the hardware, the program, and the variability of the Time-Sharing System's response time.

The economics of such a capability is another area that has only been cursorily examined. It is obvious that the cost of the hardware alone precludes the use of character recognition as a simple replacement for a keyboard console. This

is true from several points of view. First, the dollar costs of the two kinds of terminals and their interface requirements make the keyboard console more immediately attractive. Secondly, user experience and capability (given that both devices are doing the same job) would make the keyboard console the more desirable for most people. Therefore, the payoff (if one exists) of on-line character recognition lies in those areas where input is either impossible or extremely difficult to achieve through a keyboard console. This is the problem area to which we have addressed ourselves, and this is the kind of capability we believe we have attained with the present version of the recognizer. Namely, for a given individual, we can provide a larger character set tailored to his needs than that available through a keyboard--that is, the character set is made up of his own choices from a much larger set. More importantly, for the first time the position- and size-independent nature of the recognizer permits a user to input complex two-dimensional notations of practically any discipline for computer processing. This job cannot be done easily by any other method. It is here that future activity must take place in the development of systems and applications that require free-form, two-dimensional character input.

Our recognizer is far from perfect, and we intend to continue improvements and explorations into other techniques and approaches, as well as attempting to make meaningful use of our current capability in the near future.

6. BIBLIOGRAPHY

1. Bernstein, M.I. "Computer Recognition of On-Line Hand-Written Characters." RM-3753-ARPA, The RAND Corporation, Santa Monica, California. October 1964.
2. _____. "An On-Line System for Utilizing Hand-Printed Input" TM-3052. System Development Corporation, Santa Monica, California. July 1966.
3. _____. "A Method for Recognizing Hand-Printed Characters in Real Time" Proceedings of the IEEE Pattern Recognition Workshop - 1966. (In press).
4. _____. "An On-Line System for Utilizing Hand-Printed Input: A Progress Report" TM-3052/001, System Development Corporation, Santa Monica, California. December 1967.
5. Brown, R. M. "On-Line Computer Recognition of Hand Printed Characters." IEEE Transactions on Electronic Computers. December 1964, pp. 750-752.
6. Dimond, T. L. "Devices for Reading Hand Written Characters." Proceedings of the Eastern Joint Computer Conference. December 1957, pp. 232-237.
7. Kuhl, F. "Classification and Recognition of Hand-Printed Characters." IEEE International Conference Record. Vol. II, Part 4, 1963.
8. Teitleman, W. "New Methods for Real-Time Recognition of Hand-Drawn Characters." Report No. 1015 Bolt Beranek and Newman, Cambridge, Massachusetts. June 1963.
9. Nugent, W. R. and L. F. Buckland. "Improved Text Editing Using Hand-Drawn Commands and Data: A Technique for RAND Tablet and CRT Display." Second Quarterly Progress Report. Inforonics, Boston, Massachusetts. November 1966.
10. Gallenson, L. "A Graphic Tablet Display Console for use Under Time-Sharing." Proceedings of the Fall Joint Computer Conference, Vol. 31. Anaheim, California. November 1967.

APPENDIX

DETAILED DESCRIPTION OF THE RECOGNIZER PROGRAM

This document describes a program for recognizing hand-printed information in real time. This program provides on-line computer users with a means for inputting two-dimensional information into a machine as simply as writing with pen and paper. Operating under the Time-Sharing System on the Q-32 computer at SDC, the program uses a RAND Tablet for input and a CRT display (rear-projected on the tablet) for output. Each user of the program builds a unique character dictionary, based on samples of his own input characters. For each user, the program currently recognizes about 100 different characters, which are chosen from a larger character set by the individual user. This document describes in detail the various segments of the character recognition program and their interrelationships. It also includes program flow charts for each of the segments; a list of special notation used; an explanation of tables used by the program; a glossary of mnemonics and abbreviations used; and an example of dictionary optimization.

1. INTRODUCTION

The following is a detailed description of the character recognition program as of February 1968. The program described runs under the SDC Time-Sharing System (TSS) on the Q-32 computer. The description has been made as machine-independent as possible, but is not necessarily independent of the system, though one need not be familiar with the intricacies of TSS to understand the program description. It is sufficient to know that all interactive I/O is carried on through a PDP-1 computer semi-independently from the main processor, the Q-32. These two computers communicate through a 16,000 (48-bit) word core bank (called Input Memory) that is directly addressable by both computers. A block of storage (1024 words) has been reserved in this core bank for refreshing the CRT display that is an integral part of the Graphic Tablet Display (GTD) console.

A special interface program has been included in the PDP-1 for the RAND Tablet. This program, called GRID, is the only one described below that operates in the PDP-1; all others operate in the Q-32 under TSS.

2. ANALYSIS

We have yet to institute formal testing of the character recognizer, although several people have successfully used the program during investigation of applications for the technique. The primary reason for not formally testing the recognizer is its continuous state of change. Corner detection has been improved, but perfection has not been attained. Intersections (cross-overs) within a stroke are now found and have proven valuable in eliminating many of the ambiguities that bothered us earlier. The present cross-over computation is time-consuming and we believe a better way can be found. The feature extraction method itself has been changed markedly and the resulting improvement in performance has shown that effort to be worthwhile. All of these things delay testing, but the major problem in testing a program such as this is designing a meaningful test. What precisely should be tested and to whose satisfaction? Should a random sample of people be chosen to build a dictionary of some chosen subset of characters? What restrictions should be placed on dictionary building, total number of samples, maximum number per character? What is acceptable performance and to whom? How should the subjects be motivated, toward high individual performance or toward "beating" the program? Who should do the testing and under what circumstances? What can be learned from such a test? We feel that it reasonable to have answers to these and other questions before launching a testing program for the present version of the character recognizer or its successors.

Though as yet not rigorously demonstrated, the original goals of this project--we believe--have been met. Namely, the program recognizes at least 100 characters for a given individual, though probably not for any individual chosen at

random. We have implemented ways of manipulating the dictionary of character definitions that allow purging unwanted portions of the dictionary, optimizing definitions for multi-stroke characters, and merging dictionaries that were created separately. As yet, we have not found a way of optimizing the multiple definitions for single-stroke characters that does not complicate the dictionary structure to a degree that makes new definition entry and searching exceedingly time-consuming.

3. NOTATION

Because there exists little, if any, standard notation in either the field of programming or character recognition, we have invented notation where we felt it lent both brevity and clarity to the presented material, and have stayed as close as possible to "accepted" representation in all other cases.

xy or p

Represent an ordered pair of coordinates; usually provided by the input. p is used when it will not cause ambiguity. When operations or functions are applied to p, they are applied fully to both x and y. Both representations may be subscripted, either to denote a particular order in the sequence of coordinates that make up a stroke, or to identify a unique pair.

dxy or Δxy or exy

Represent the signed difference between two pairs of coordinates, $x_1 - x_2$, $y_1 - y_2$. When tests or operations are performed on individual members of the pair, they are separated in the form Δx and Δy .

h

Represents the heading--as computed by the function DIRQ (Δxy)--between two points. The values are treated as signed two's complement fractions for computational purposes.

Δh

The signed difference between two headings, h_1 and h_2 . The sign indicates the direction of rotation (minus for counter-clockwise and plus for clock-wise), and the magnitude represents the amount of directional change. Note that since the computation is done on signed two's complement fractions, the largest change that can occur is 180° , indicated by a -1

<u>S</u>	Stands for stroke. In our program, a stroke is made up of the ordered set of coordinates, p , input to the program between the sequence of pen switch on and pen switch off.
<u>f</u>	Stands for feature. Usually it is only a portion of a stroke.
<u>Min and Max</u>	Used in their usual mathematical meaning. When either is applied to a stroke, S , the x and y coordinates are treated independently. Therefore, $\min(p_i)$ means $\min(x_i)$ and $\min(y_i)$; $\max(p_i)$ has a similar meaning.
<u>R(Z)</u>	The minimum rectangle surrounding the set of points, Z , which may be a stroke, S , or a subset of the stroke. Usually that subset that constitutes a feature $f(Z)$ is obtained by computing $\min(Z)$ and $\max(Z)$ and is a pair of ordered pairs (x_{\min}, y_{\min}) , (x_{\max}, y_{\max}) .
<u>$\bar{R}(Z)$</u>	The center of the minimum rectangle $R(Z)$. It is an ordered pair x, y . It is computed as $1/2(x_{\min} + x_{\max})$, $1/2(y_{\min} + y_{\max})$.
<u>Size (A)</u>	An ordered pair of differences, Δxy , specifying the size of an entity A . It is usually computed from $R(A)$. The notation used in the flow charts is $\text{Size}(R(A))$. It is computed as $\Delta x = x_{\max} - x_{\min}$, $\Delta y = y_{\max} - y_{\min}$. Note that both Δx and Δy are ≥ 0 .
<u>Ch</u>	Stands for character. Ch^{\emptyset} is used to specify an output character, and may only be a legitimate member of the output character set (the set from which the user defines his input). Ch^I is used to specify an input character, and is a collection of from one to n strokes, $(S_0, S_1 \dots S_n)$.
<u>σ</u>	Represents output from the ANALYZER routine other than a Ch^{\emptyset} . Five values have been used. They are: σ_{\circ} = a stroke with too many features or (in the SAMPLE routine) a Ch^I with too many strokes.

- σ_1 = a stroke or character not found in the dictionary; thus, an undefined character.
- σ_2 = a stroke or character in the dictionary that ends at an intermediate node, has no $\text{Ch}\emptyset$ attached, and is, therefore, an intermediate, undefined character. For example, if the first sample provided to SAMPLE were a four-stroke M and the dictionary were empty, the first three strokes entered would have a σ_2 appended to them; the fourth stroke would have the $\text{Ch}\emptyset$ "M" appended as its definition.
- σ_3 = end of strokes or vacuous stroke.
- σ_4 = an invalid stroke, defined as one for which $\Delta x + \Delta y \geq 20$ for any adjacent pair, p_i, p_{i+1} .
- σ_5 = an empty position used for initialization purposes.

A(S)

Represents the ANALYZER output for S. A(S) may have 0 as an intermediate value, but its final value is always either $\text{Ch}\emptyset$ or σ .

4. TABLES

Various tables are referred to in the following description (both in the prose and in the flow charts). In the context used here, each table is made up of entries and each entry is referred to using the indexed table name. Each entry may be made up of items. These items are referred to using the item names, and every item in the same entry has the same index as that entry. In tables whose entries contain items, a reference to an entry implies the collection of items in that entry. For example, in table GRØUP, each entry contains the two items LSort and LGroup. The statement "Clear GROUP_i " means that both LSort_i and LGroup_i are set to zero. In addition, the statement "Clear Table GROUP" means that both items LSort and LGroup in every entry are set to zero.

Table 1 is a list of the tables used in this program. It includes the names of the items in each table entry and the number of entries in each table. In order to conserve core space, the table SØRT overlays DB, and GRØUP overlays PTS. This causes no conflict since the program is not interactive when those tables (SØRT and GRØUP) are in use.

Figure 1, which shows the relationship between the various routines of the recognition program, also contains the list of tables that each routine uses, and--following the table name in parentheses--the indices used by that routine (if any) to refer to the table entries.

Table 1. Tables, Entries, and Items Used in the Recognition Process

<u>Table (Entry) Name</u>	<u>Items</u>	<u>No. of Entries</u>
IMB	x,y,M,C	1024
DB	x,y,M,C	1024
PTS	X,Y,I,Ct,h, h (Q overlays Ct in XØVER)	300
STRK	F,E,G,A(s),D,R(S),R̄(S),R(f)	15
DICT	F,E,G,Def,Rc,NLink,SLink	512
DICTE	DEQ,DGroup,DGLink,DMark,DTØ	512
DICT*	F*,E*,G*,Def*,Rc*,NLink*,SLink*	512
SØRT	SØRTL,SØRTR	1024 overlays DB
LEVEL	LThis,LLast	15
GROØP	Ch,GChain,GNLink,GSLink	150 overlays PTS
LTAB	LSort,LGroup	15
ATAB	AI,Abeg	5
CTAB	CI,Cbeg,Cend	10

5. PROGRAM DESCRIPTION

5.1 DATA FLOW

Ignoring those portions of the implemented program that are used for testing and debugging, we shall describe the sequence of events that take place during the execution of the program. The flow of data through the system, including both input data and control signals, will be described.

The over-all functioning of the system involves three hardware units, plus the software of the time-sharing system. The three hardware units are: (1) the PDP-1 computer, which interfaces and buffers all interactive input and output; (2) the Input Memory, a 16 K core storage module of 48-bit words that is directly addressable by both the PDP-1 and Q-32 computers; and (3) the Q-32 computer, a large (65K 48-bit word), fast (2.5-usec cycle time), general-purpose digital computer on which TSS runs.

The Input Memory can be directly read by an object program running under the Time-Sharing System in the Q-32, but only the TSS supervisor may write in Input Memory. Because there are no user programs running in the PDP-1, there are no restrictions on its use of Input Memory. (Note that GRID is considered to be a system program, not a user program.)

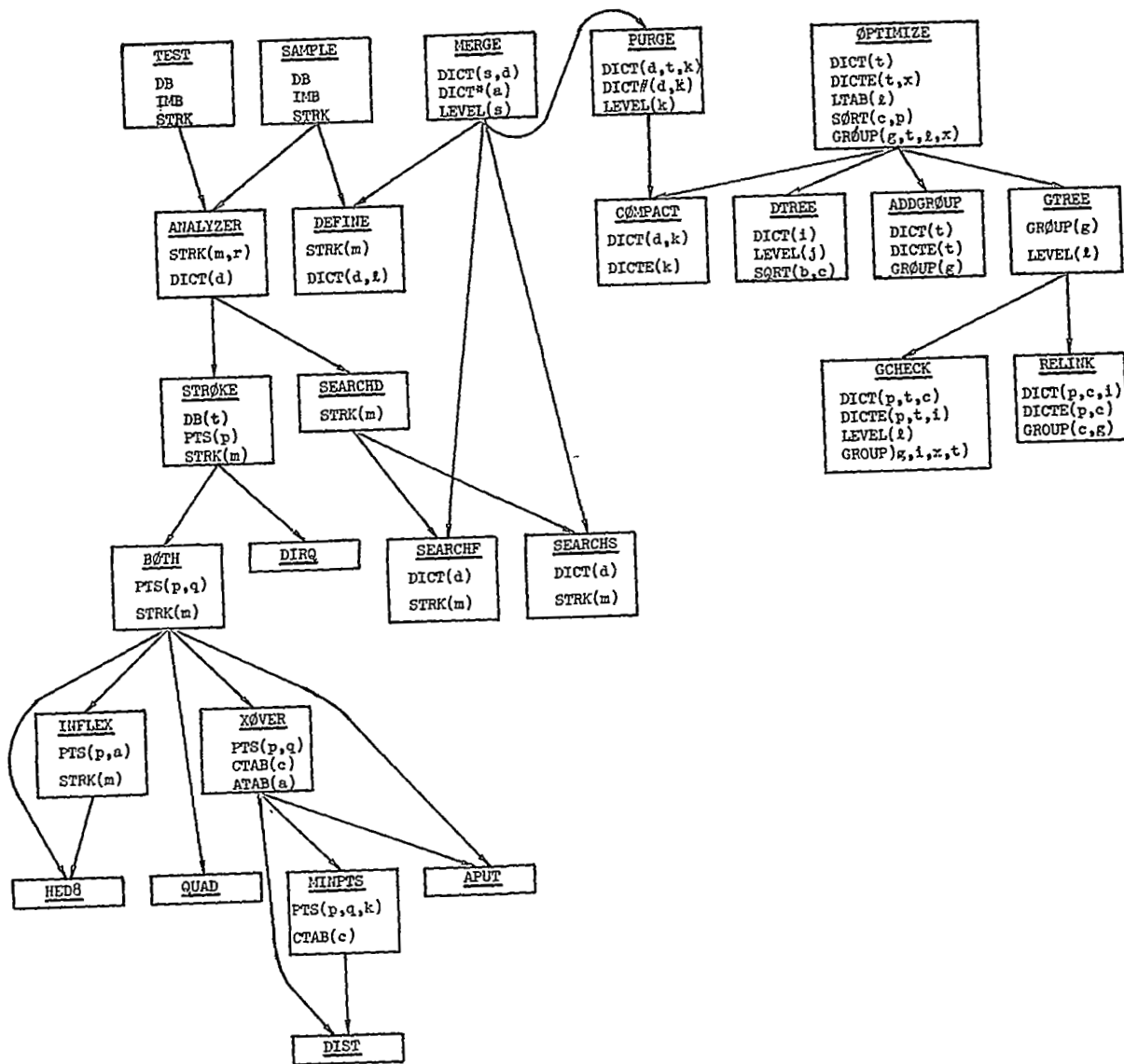


Figure 1. Character Recognizer Routine Relationships

The Graphic Tablet Display Console, around which this program is constructed, interfaces with the above hardware as follows: the RAND Tablet (Grafacon 1010A) used for input is connected to the PDP-1 through a hardware interface. Each time an input is ready at that interface, the PDP-1 is interrupted. These interrupts occur at two different rates: every 30 ms when the tablet's pen switch is off, and every 4 ms when it is on. This pen switch on rate is variable between 1 ms and 16 ms. The 1-ms rate cannot be handled by the PDP-1; samples greater than 4 ms apart would not provide adequate data.

The CRT display for the console is directly connected to Input Memory through another hardware interface. A block of 1024 words of Input Memory is reserved in a fixed place for the refresh buffer of the display. The interface reads from this buffer and "paints" the contents on the display (continuously) at a rate of approximately 32 frames per second. The display refresh may be turned on and off manually. When on, the contents of the 1024-word buffer are shown on the display without regard to the condition of other parts of the system.

To start the character recognition program, it is loaded for execution under TSS via a teletype console in the same way as any other user's program. The program then queries the user as to his intent. For purposes of discussion, assume he wishes to use the tablet for input. He indicates this intent. The program* (running in the Q-32 under TSS) then requests that the tablet (in reality, the GRID program of the PDP-1, which is a system routine, not a user program) be attached to this user as a private device. This insures that all inputs from the tablet are directed only to the user's program. The GRID program at this point is most likely to have SW1 set to IGN (see Figure 2), though it may have been left set to TB, but that will not affect program execution. The program then generates a display image in Q-32 core consisting of three pushbuttons, labeled "DRAW", "SAMPLE", and "TEST." Only the latter two concern us here. This image block also contains the appropriate control words to set the GRID program for use--that is, to set SW1 to TB and assign the first relative location in the buffer where GRID may store its inputs. This block of Q-32 core (which we shall refer to as DB) is transferred to the Input Memory display buffer (which we shall call IMB) by a call to the TSS Supervisor. When TSS returns control to the user's program, it then requests input from GRID and goes into "wait" status through another supervisor call. This means that the object program in the Q-32 will not be run again until the GRID program informs the Q-32 that input has been completed and it is giving up control.

With GRID in control, the user now positions his reflected pen position spot over the "SAMPLE" Button and presses hard enough to turn the pen switch on;

*This routine is not part of the recognition program itself, and thus is not documented here.

he then releases the pressure and the switch goes off. GRID has placed the input points in IMB, so that they are now a part of the display image. The Q-32 only allows space for one input point by GRID. When it finds that the buffer is full, it calls TSS to take the user out of "wait" status; GRID then essentially goes to wait status itself. (If there were more space in IMB, control would have been returned after a time delay on pen up of 1/4 sec.)

The user is informed that control has returned to the Q-32 by the lack of the moving spot reflecting his current pen position. Note that it is possible to enable GRID to accept inputs without putting the Q-32 program into "wait" status, but we have not tried this, even accidentally. If this were done, the two programs (GRID and the user's program in the Q-32) could get out of synchronization and both end up waiting for each other, thus effectively killing the program.

The Time-Sharing System next takes the user's Q-32 program out of "wait" status. The program reads the pen input from Input Memory, decodes the coordinates in terms of button position, and either rejects or accepts them. If the input is rejected, the program simply goes back to the beginning, rewrites Input Memory, calls GRID and waits. If the inputs are accepted, the program named in the button pushed (in this case, SAMPLE) is called and control passed to it.

SAMPLE generates its initial frame in DB. Having divided up the buffer block appropriately, it sets the time delay (TD) to 2 seconds, turns the smoothing flag (SMFLG) on, calls the supervisor to write DB into IMB, calls GRID, and goes into "wait" status.

The user next sees a new display. He may then push a button to change the keyboard displayed, or draw a character in the writing area. GRID reacts the same way as above. Assume that the user has drawn a character, say a "|". The user should then wait for some response from the Q-32 program to show on the display before he proceeds. If he inadvertently starts to draw or push a button before GRID has given up control, he will add information to the input that he has not intended and may--as a result--unknowingly add a meaningless character definition to the dictionary. When control is returned to SAMPLE by the system, it checks the validity of the input as a function of both the setting of CHSW, and the position of the initial and final coordinates of the first input stroke. (A stroke is defined as all of the input occurring between pen switch on and pen switch off).

Finding a valid stroke, SAMPLE calls ANALYZER. The stroke is analyzed and the resultant feature string searched for in the dictionary. Note that when SAMPLE is in control, ANALYZER processes all strokes before returning, on the assumption that all of the input belongs to one character. Since the user's dictionary is empty, the input will not be found, SAMPLE adds the special character "v"

to the display image, relocates the INKORG so that the inputs will not be erased, sets the CHSW to U, writes DB in the IMB, calls GRID and waits. The user must now "push" one of several buttons. If he draws another character or pushes the wrong button, his input will be erased and the program will wait for another input.

After completing the dictionary, the user makes an input at coordinate 0,0. This, in essence, causes SAMPLE to return control to the program that called it--the control program of three pushbuttons. If the user were to make an input at 0,0 in this control program, he would return control to the teletype.

The difference between SAMPLE and TEST is that there are no invalid inputs to TEST. Multi-stroke inputs are not treated as a single character.

The ability to erase or replace an output character from the display is not crucial to the objectives of the TEST. Its main function is to test the user-constructed dictionary. Replacement and erasure were incorporated to provide a more realistic usage environment, and to create a better demonstration vehicle. Generating an output character of the same size and at the same position as the input character aids in creating the proper atmosphere. These capabilities were added to TEST in order to solve some of the known problems that will occur when the character recognizer is used as a tool for an actual problem solution.


All of the information concerning output character position, size, and location in the output buffer is placed in DLIST, which is a simple linked list, external to the actual buffer. DLIST could have been implemented in several ways. The linked list was a design choice.

If the user then pushes "TEST", control will be turned over to the TEST program. TEST sets up DB in two portions: (1) a simple linked list to minimize wasted space in displaying arbitrary characters in 5 x 7 dot matrix form, and (2) a block reserved for "ink".

Assume that the user draws

CAT

This takes five strokes. When GRID returns control to TEST, the inputs are moved from IMB to DB, and ANALYZER is called. ANALYZER returns control to TEST if one of several events has occurred: (1) all of the strokes in DB have been processed; (2) the matching feature string in the dictionary has no successor stroke appended to it; (3) the next stroke processed does not match any of the successor strokes or it cannot match a first stroke; (4) the input has too many features and is thus considered a "scrub" or erasure, or it has

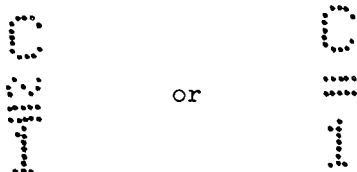
a noisy point and is considered invalid. One of the results of this kind of interactive control is that a non-recognized stroke can change the "meaning" of the remaining strokes from that intended. Each time ANALYZER returns control to TEST, TEST adds or deletes characters in DB; only when all strokes are processed does the result appear on the user's display, replacing his hand-drawn input. If all goes well, the user will see  appear at once on

the display, although each character was placed in DB as it was recognized and returned by ANALYZER.

To illustrate what can happen if a stroke is not found, assume that the user draws the following:



on the tablet in the order, C, A, -, -, I, and the ANALYZER could not find the "A" of the "A" in the dictionary. The user would see either



depending on where the center of the "A" is with respect to the two strokes that are not taken as an "=". The first of these two illustrations shows the most likely case.

5.2 RESPONSE TIME

We have been able to provide instantaneous response in the one critical area required--namely, feedback from pen inputs to the display so that "ink" flows from the "pen" in real time.

Response time for the user upon completion of input is a function of many variables. Naturally, the more interactive users there are on the system, the longer the delays. In particular, the more people using tape and disc, the longer the delays. Also, the more input there is from the tablet, the longer are the potential delays. The effect of this factor is hard to measure because of the other variables, but if the processing takes more than one quantum of interactive user time (currently about 600 ms), the user is swapped out and is at the bottom of the interactive queue for one cycle.

In general, when there are less than 20 other users on the system, response is acceptable (though not instantaneous). When there are more than 25 users on the system, response is slow enough to be annoying; when there are more than 30 users on, response can become intolerable.

If the SDC Time-Sharing System had provisions for priorities among the interactive users, some of these annoyances could be lessened, but it doesn't and so from time to time we will be annoyed.

6. DESCRIPTION OF PROGRAM SEGMENTS

6.1 GRID SUBROUTINE

GRID is a program that runs on the PDP-1 and provides the interface between the RAND Tablet and the Q-32 Time-Sharing System.

All communication between GRID and the calling program takes place via the Input Memory Buffer (IMB). The calling program supplies the variables (SMFlg, TD, F and INKLOC) in the control words and GRID places the tablet input data (ink) in the buffer beginning at the location specified in INKLOC. The format of these inputs is shown in Figure 2.

A flow chart of the GRID subroutine is shown in Figure 3. It is not essential that this routine start with SWI at IGN, since the time-out on TD guarantees that SWI cannot remain indefinitely in the TB position if any inputs have been made, regardless of the state of the Q-32 program. Note that the PDP-1 is interrupted at two different intervals: at 4-ms intervals when psw (pen switch) is on, and at 30-ms intervals when psw is off.

SW2 is set when psw is first detected. This saves both time and space, because of the way the PDP-1 reads Input Memory. The smoothing algorithm (enabled when SW2 is set to "on") merely replaces the oldest xy with the one just read, then computes the average xy as output. The index j is operated as a ring counter, modulo 8.

The filter factor F is another of the variables supplied by the calling program. A value of 3 is used for the filter. It was arrived at empirically for drawing very small characters. Ns is the number of strokes input. Until recently, Ns has not been implemented properly, thus it is used only by time-out testing in GRID and not by the Q-32 routines. Another function that could have been included, had space been available (time was no problem), is the computation of the minimum rectangle surrounding each stroke, R(S). This would have saved some time in the Q-32 programs, at the cost of buffer space in the INK area.

6.2 SAMPLE SUBROUTINE

The SAMPLE subroutine is used to build and test dictionaries in a one-character-at-a-time mode. This program allows only one action at a time--drawing a character (multi-stroke characters are allowed) or pushing a button.

Figure 4 shows a layout of the display the user sees when this program is called. The user can then select one of five keyboards. The five keyboards contain (1) digits, brackets and relationals; (2) upper-case Roman letters; (3) lower-case Roman letters; (4) punctuation and special marks; (5) Greek letters. The keyboards are shown in Figure 5 as they appear to the user. The maximum number of characters per keyboard is 26.

A flow chart of the SAMPLE subroutine is shown in Figure 6. Those parts of the flow chart concerned with the detailed control of the Display Buffer indicate when various strokes are left and when they are erased. The buffer is allocated as shown in Figure 7.

Communication between SAMPLE, ANALYZER, and DEFINE is through the inputs in the DB, the STRK Table, and a set of standard global communication registers plus, of course, the computer's accumulator.

When an input is found in the dictionary as a defined character, the output character replaces the input at the same place on the display at approximately the same size. If the input is not defined, it is permitted to remain and one of two special characters, "v" or "2", is output at the ILØC (a position on the display surface) (see Figure 2).

The CALL GRID AND WAIT is a Time-Sharing System dispatcher call. The call could be given without a wait, but there is nothing that needs to be done in the interval; also synchronization between the two programs would be more difficult.

Although not shown in Figure 6, there is a small master program to which SAMPLE and TEST (see below) exit when the input stroke is found to be on the 0,0 coordinate of the tablet.

6.3 TEST SUBROUTINE

TEST allows the user to test a dictionary in a multi-character, interactive mode. In addition, it provides two editing features: replacement of an existing character with an input, and erasure of one or many characters with one scrub. A flow chart of the subroutine is shown in Figure 9.

The mode flag is set to "test" so that ANALYZER does not assume that all of the input strokes constitute one character.

The Display Buffer for TEST is organized as shown in Figure 8.

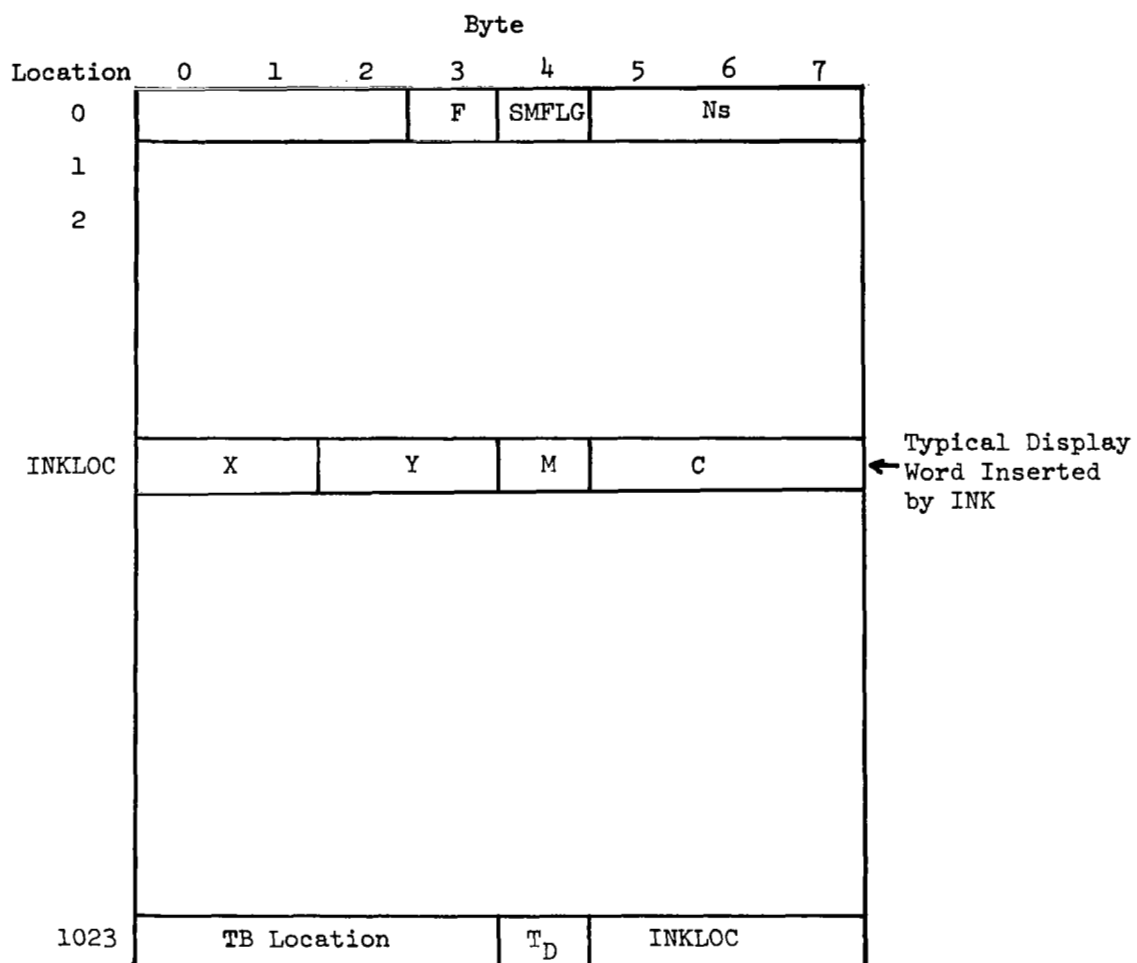


Figure 2. Storage Map for Display Buffer (DB) and Input Memory Buffer (IMB)

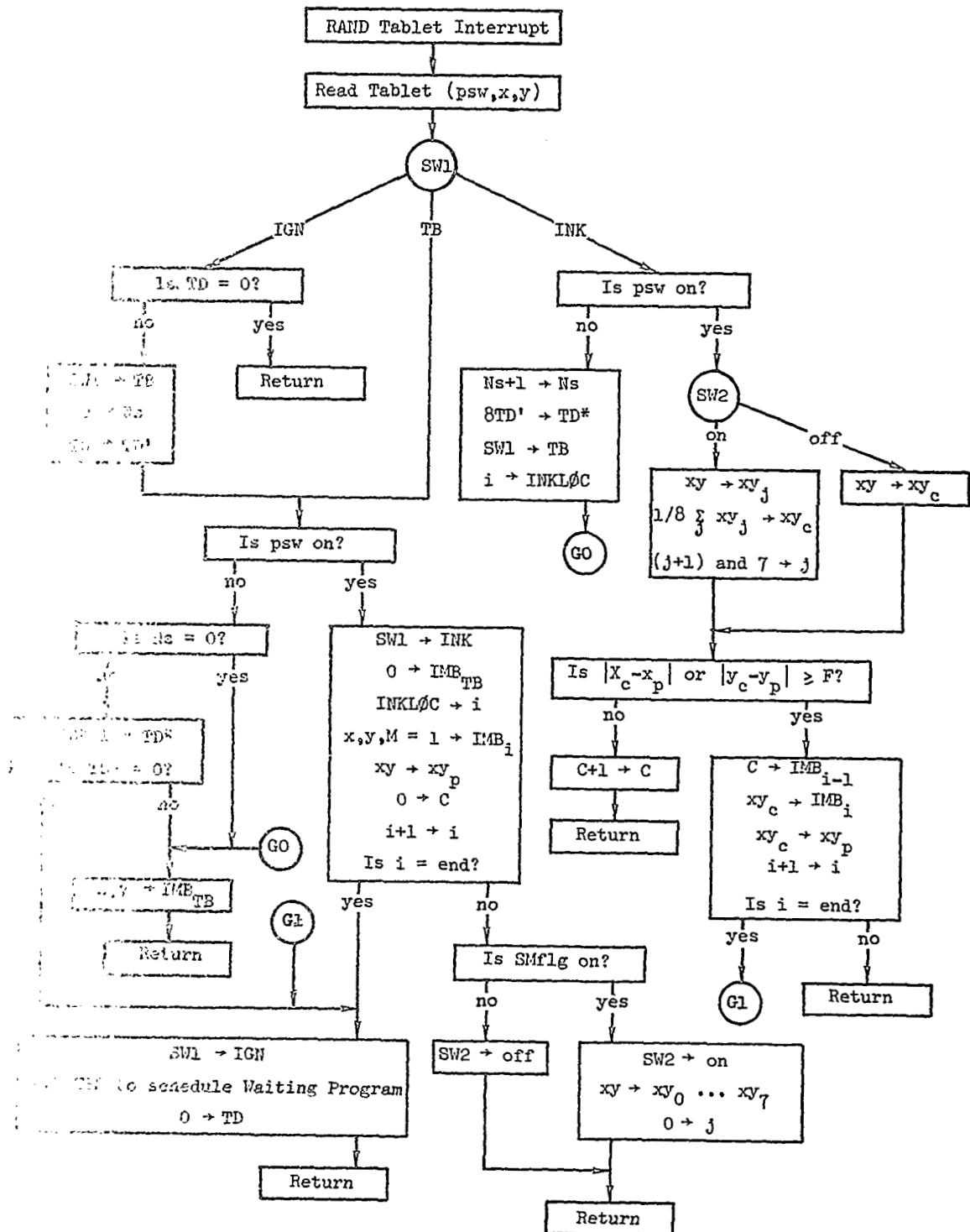


Figure 3. GRID Subroutine

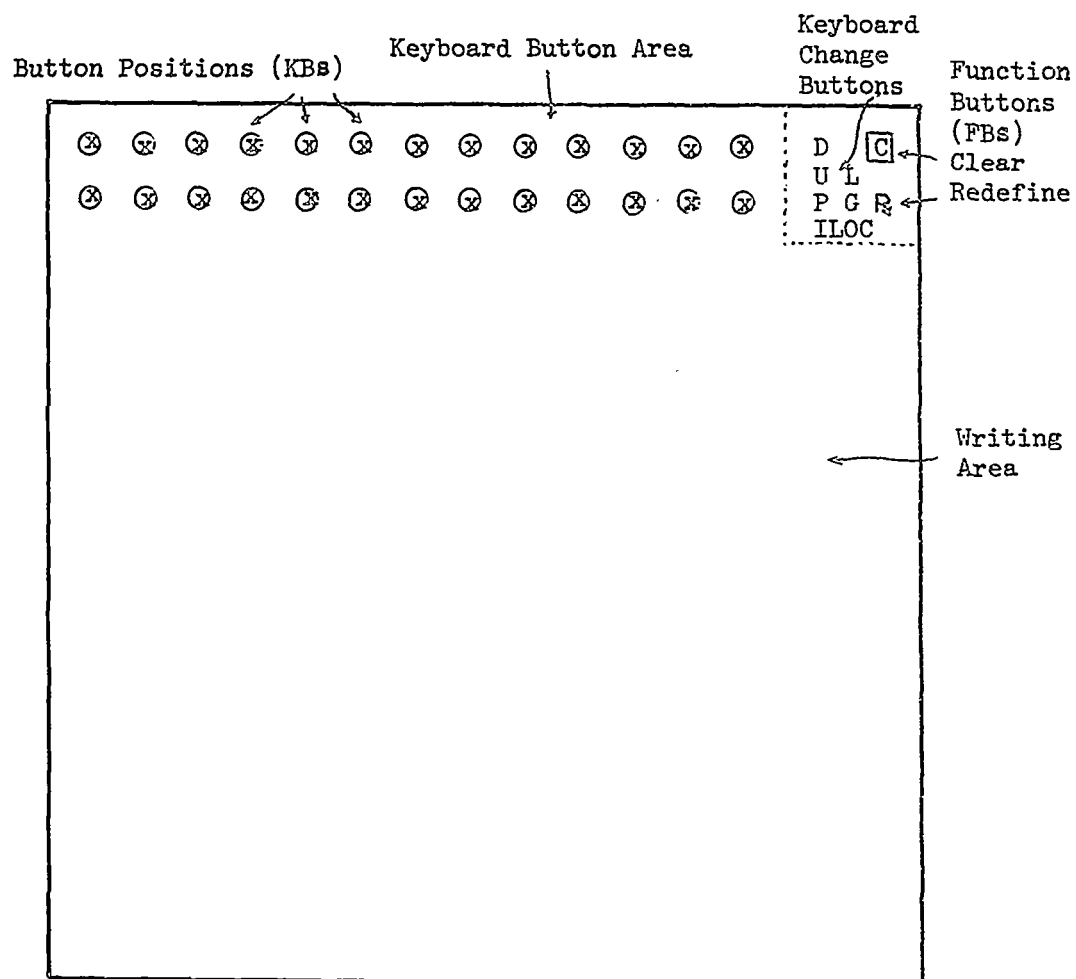


Figure 4. Display for SAMPLE Program

0	1	2	3	4	5	6	7	8	9
<	≤	≥	()	[]	{	}	

a. Digits, relationals and brackets

A	B	C	D	E	F	G	H	I	J	K	L	M
N	O	P	Q	R	S	T	U	V	W	X	Y	Z

b. Upper-case Roman letters

a	b	c	d	e	f	g	h	i	j	k	l	m
n	o	p	q	r	s	t	u	v	w	x	y	z

c. Lower-case Roman letters

.	,	!	:	;	?	/	\	+	-	*	/
=	%	<	>	%	&	~		_	^	°	°

d. Punctuation and special marks

α	β	γ	δ	ε	ζ	η	θ	ι	κ	λ	μ
ξ	π	ρ	σ	τ	υ	φ	χ	ψ	ω		

e. Greek letters (and other special marks)

Figure 5. Character Subset Keyboards

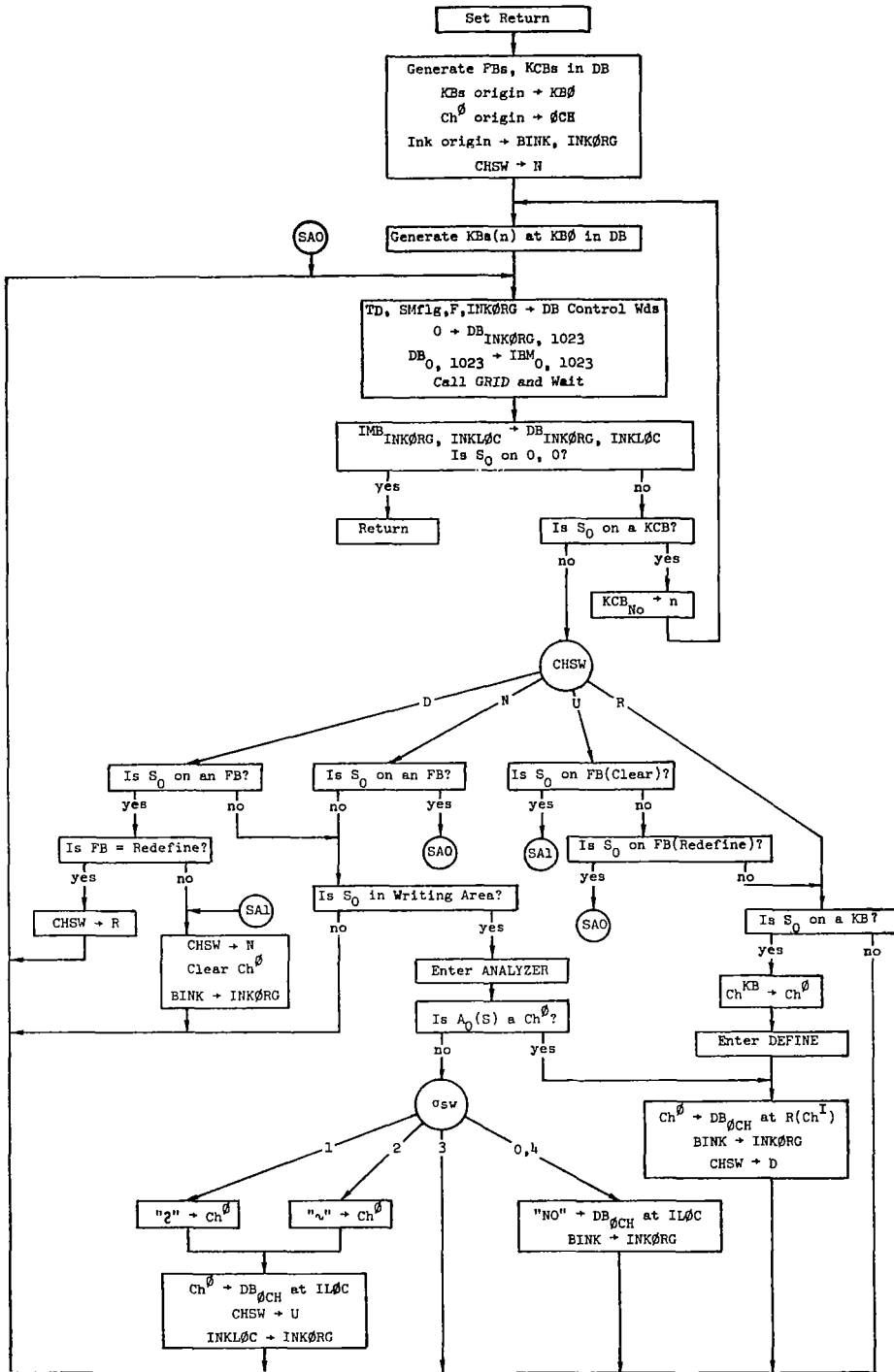


Figure 6. SAMPLE Subroutine

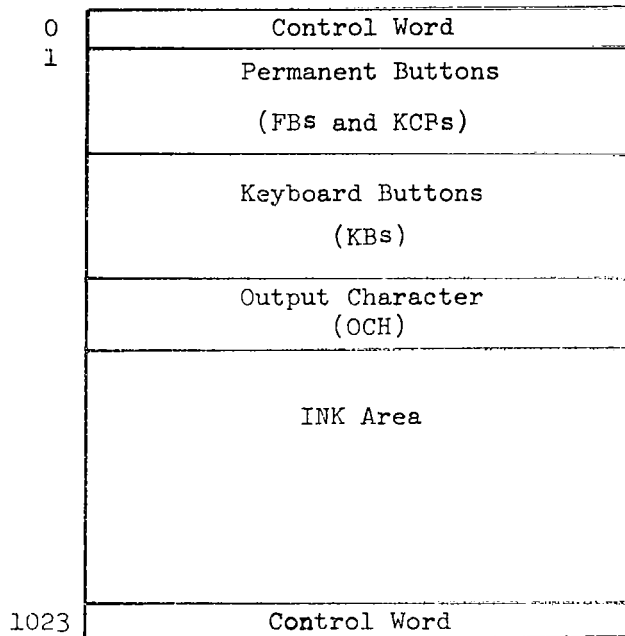


Figure 7. Display Buffer Allocation for SAMPLE

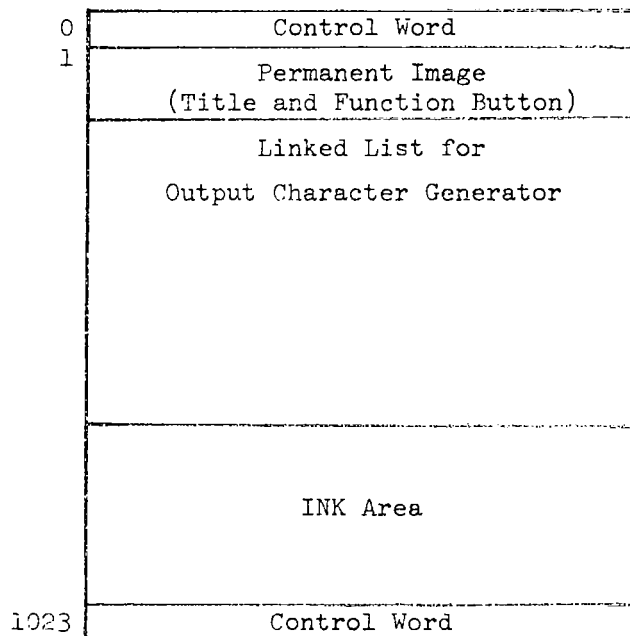


Figure 8. Display Buffer Allocation for TEST

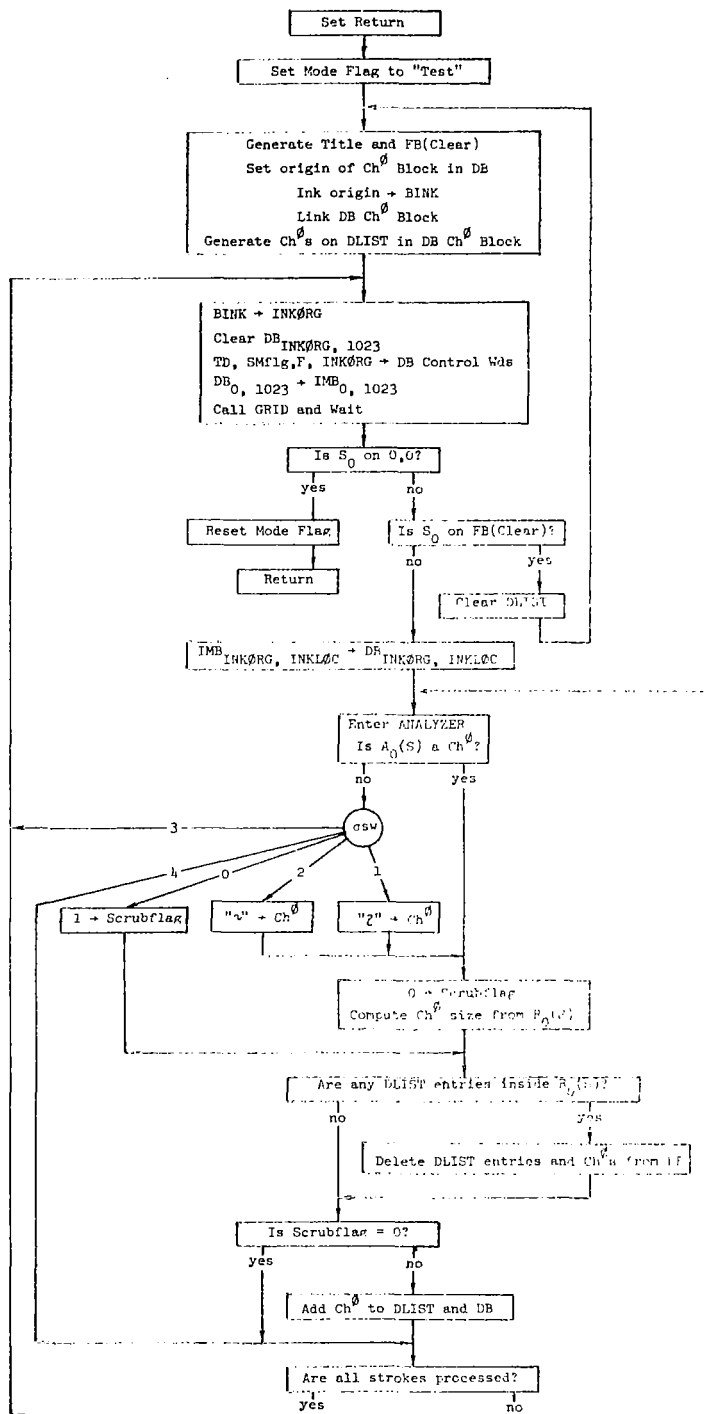


Figure 9. TEST Subroutine

The Display List (DLIST) is a linked list of output characters and other information required for physically locating the individual characters on the display surface.

Each element contains the following:

1. Location of the lower left-hand corner of the output character.
2. Size ($R(\text{Ch}^I)$).
3. A pointer into the Display Buffer to the origin of the points forming the output character.
4. The output character code.
5. A link to the next item (0 indicates the end of the Display List).

The limited ink space in the Display Buffer (actually the IM Buffer) allows input of 6 to 8 small characters, 3 or 4 medium-sized characters, 1 or 2 large characters, and 1 or less very large characters. (This limitation or extra-large characters is a problem.)

When an output is passed to TEST from ANALYZER, the display list is searched to see if any characters are to be deleted. The test is performed by computing the center of the character on the DLIST and comparing to see if it lies within the minimum rectangle surrounding the input, $R(\text{Ch}^I)$. All characters for which this is true are deleted from the DLIST and DB.

Although the program currently outputs one of two special characters on the display ("v" or "2") when an unknown character occurs, it may actually be preferable to do nothing, that is, to ignore the input. For testing and debugging purposes, though, these special characters have been of value.

6.4 ANALYZER SUBROUTINE

ANALYZER is--in reality--two programs: one is coupled to SAMPLE and the other to TEST. Each part could have been included as part of those routines, or coded as separately callable subroutines, but it was more efficient to have but a single routine. The distinction between the two parts or functions of ANALYZER is obvious from a cursory examination of Figure 10.

In order to guarantee that the program executes the proper function, the variable *tmore* is initialized to zero and is maintained at that value whenever ANALYZER completes the analysis for a set of strokes in the TEST mode. Therefore, on an initial entry with a new set of strokes, the function is differentiated by examining the mode flag whose setting and resetting is completely under control of TEST.

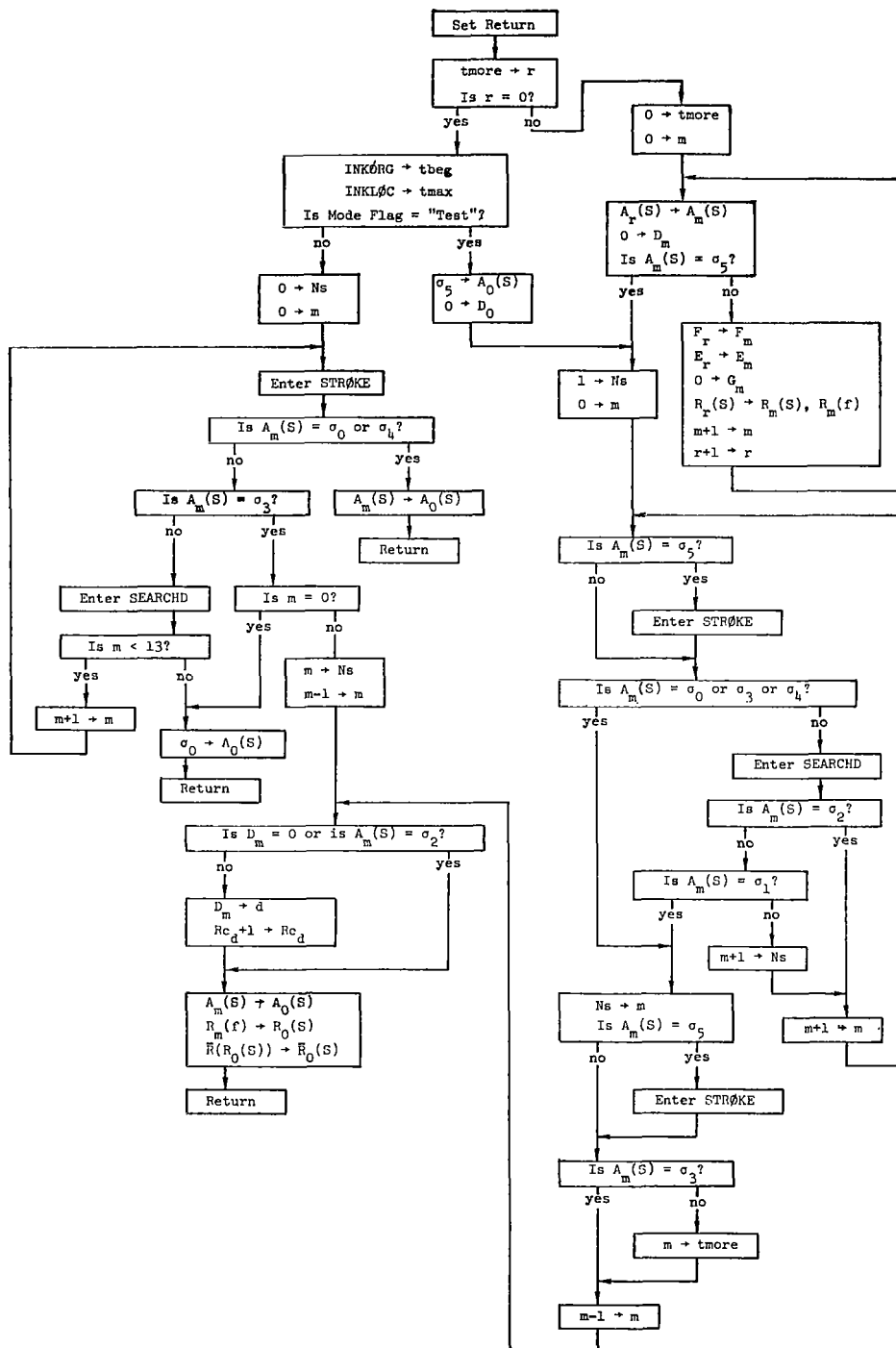


Figure 10. ANALYZER Subroutine

That part of ANALYZER associated with SAMPLE assumes that all of the strokes to be processed belong to a single input character. The program successively processes each stroke until the end of input (σ_3) occurs or until an invalid stroke (as defined in the STROKE subroutine) occurs. ANALYZER builds a table of outputs (called STRK) from the input strokes, with each entry containing the following items: F_m and E_m (the feature string for the stroke); G_m (the geometric relationship between the mth stroke and its collected predecessors); $A_m(s)$ (the result of dictionary searching or other ANALYZER results) and D_m (the dictionary definition for the mth stroke and the successor link from that entry); $R_m(s)$ and $R_m(f)$ (the minimum rectangle surrounding the stroke and the rectangle surrounding all strokes through the mth one); and--as a separate item--Ns, the number of strokes in the character.

That part of ANALYZER used by TEST is a bit more complex. Rather than being controlled by the inputs, it is controlled by the content of the dictionary. It therefore can only determine when to output a character to TEST as a result of searching the dictionary. Thus, it must be able to back up to the last legitimate character (a Ch^0) that it found, in some cases. STRK table also contains the output of the analysis. Control is returned to TEST each time ANALYZER has found (1) a node in the dictionary that has a Ch^0 as its definition and no successor link, or (2) an intermediate undefined dictionary node (σ_2) not preceded by a Ch^0 in the output table and no match on the successor stroke, or (3) can find no match for a stroke at all (σ_1) or (4) a scrub stroke (σ_0 --anything with too many features) or (5) the end of input (σ_3). Invalid strokes (σ_4) are ignored. All of the pertinent output data, $A_m(s)$, $R_m(f)$ and $\bar{R}_m(f)$, are placed in the first table entry before control is returned to TEST each time.

6.5 STROKE SUBROUTINE

STROKE processes one stroke at a time (see Figure 11). It clears the mth entry of the STRK table, and sets $A_m(s) = \sigma_1$ (undefined) and $A_{m+1}(s)$ to σ_5 (empty) as initial values. It then determines if the index pointing to the inputs is pointing at the beginning of a stroke ($M=1$ in DB_t); if not, it indexes on until it finds the beginning of a stroke or the end of input. When the beginning of a stroke is located, the xy coordinates and the rejected point count are transferred to the PTS table. Each entry of PTS contains the items X, Y, Ct (the x and y coordinates and rejected point count); h (the heading reduced to one of 32, between adjacent points); Δh (the difference between adjacent headings); and I (a marker for corners and other path features). After transferring the points from DBUFR, it determines whether or not the stroke is valid (it is invalid (σ_4) if the sum of the absolute value of the difference

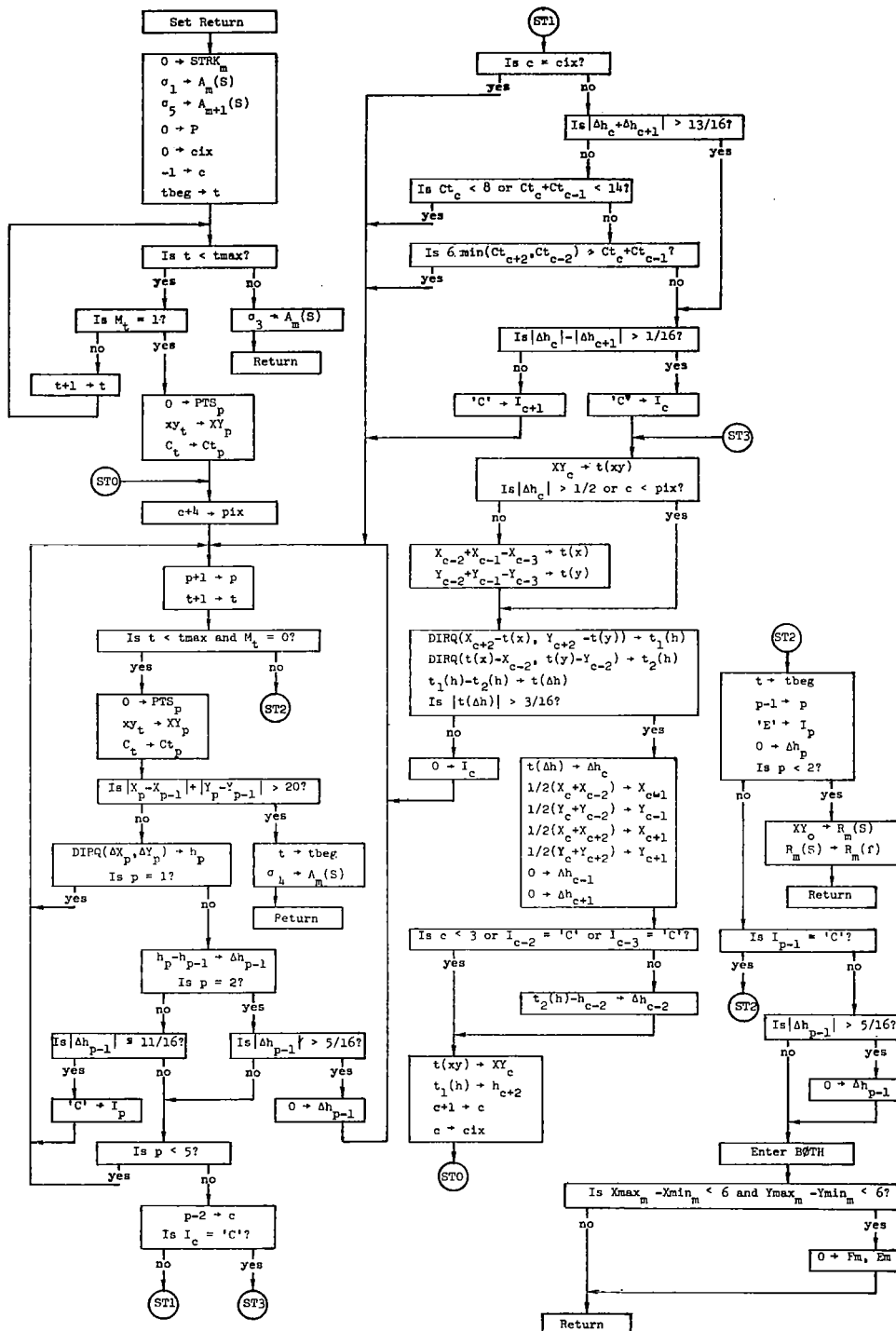


Figure 11. STROKE Subroutine

between adjacent coordinates is greater than 20), and if it is, locates any existing corners. Upon completion of corner detection, it tests for a hook that may be eliminated from the end of the stroke and enters the subroutine BØTH to continue the feature extraction.

6.6 BOTH SUBROUTINE

After initializing the indices p and q and fc (the feature count), BOTH (see Figure 12) enters INFLEX (see below) to determine if there is an inflection point between the beginning of the stroke and the first corner, or the end if no corners are present. If k (a parameter set by INFLEX) equals zero, the amount of curvature was below the threshold for a straight line, and BØTH outputs a pair of feature codes based upon a table (see Figure 12) that describes the segment (part of the stroke in question) in terms of the numbered areas (see Figure 13) occupied by the end points. If c is non-zero, an inflection point has been detected and marked, or if the curvature ($|\Sigma|$) is insufficient, no test will be made on the segment for an intersection and the area features will be generated.

INFLEX computes the minimum rectangle surrounding the segment, $R_m(f)$ and from it BØTH computes the dimension of the diamond center area, (see Figure 14). If an inflection point is marked ($c=1$), the "no output" occurs (except for the occurrence of the inflection point) while the stroke or segment is in the center diamond. On the other hand, if no inflection point is found and the total curvature in the segment or stroke is great enough, XØVER is entered (see below). It looks for closed loops and generates the appropriate feature string depending upon what it finds. Because the curvature is high and there is no inflection point, no center diamond area is computed in XØVER. Upon return, if the stroke is not completely processed, a "C" (denoting the occurrence of a corner) is concatenated to the feature string and the feature count is tested. (This is the only reason in the current program for processing less than the full stroke.) If the feature count is too large, the minimum rectangle for the stroke is computed, the output code is set to σ_0 , and a return made to the calling routine. If there is still room for more features in the string, the remainder or next segment of the stroke is processed.

6.7 INFLEX SUBROUTINES

INFLEX (see Figure 15) not only detects and marks inflection points in the path, but also computes the minimum rectangle around the segment or stroke; if it is a subsequent segment, INFLEX generates the geometric relationship feature between the current segment and the collection of predecessors in this stroke. In detecting inflections, initial limits on curvature are set at $3/8$ for a clockwise rotating stroke, and $-3/8$ for a counter-clockwise

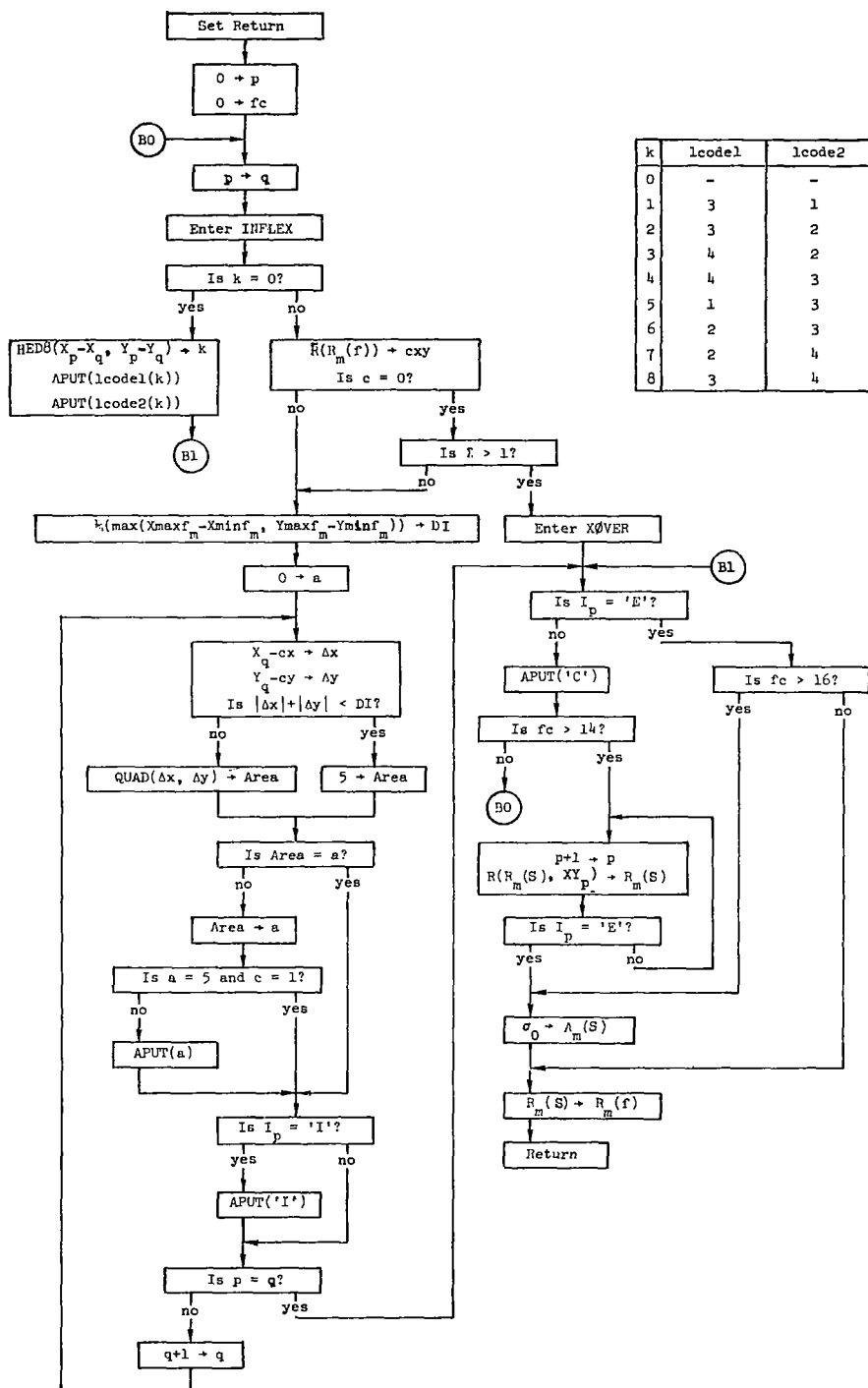


Figure 12. BØTH Subroutine

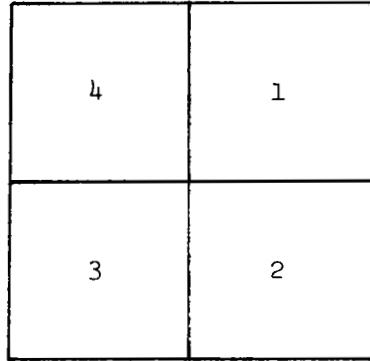


Figure 13. Segment End-Point Areas

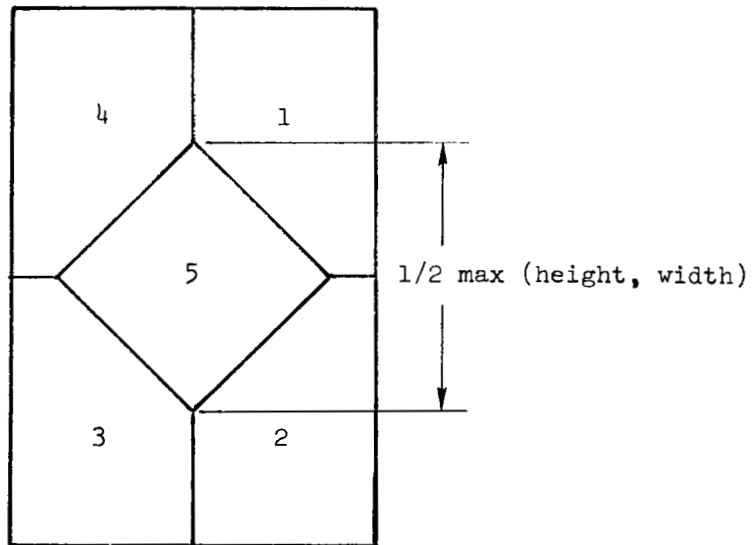


Figure 14. Segment Feature Areas

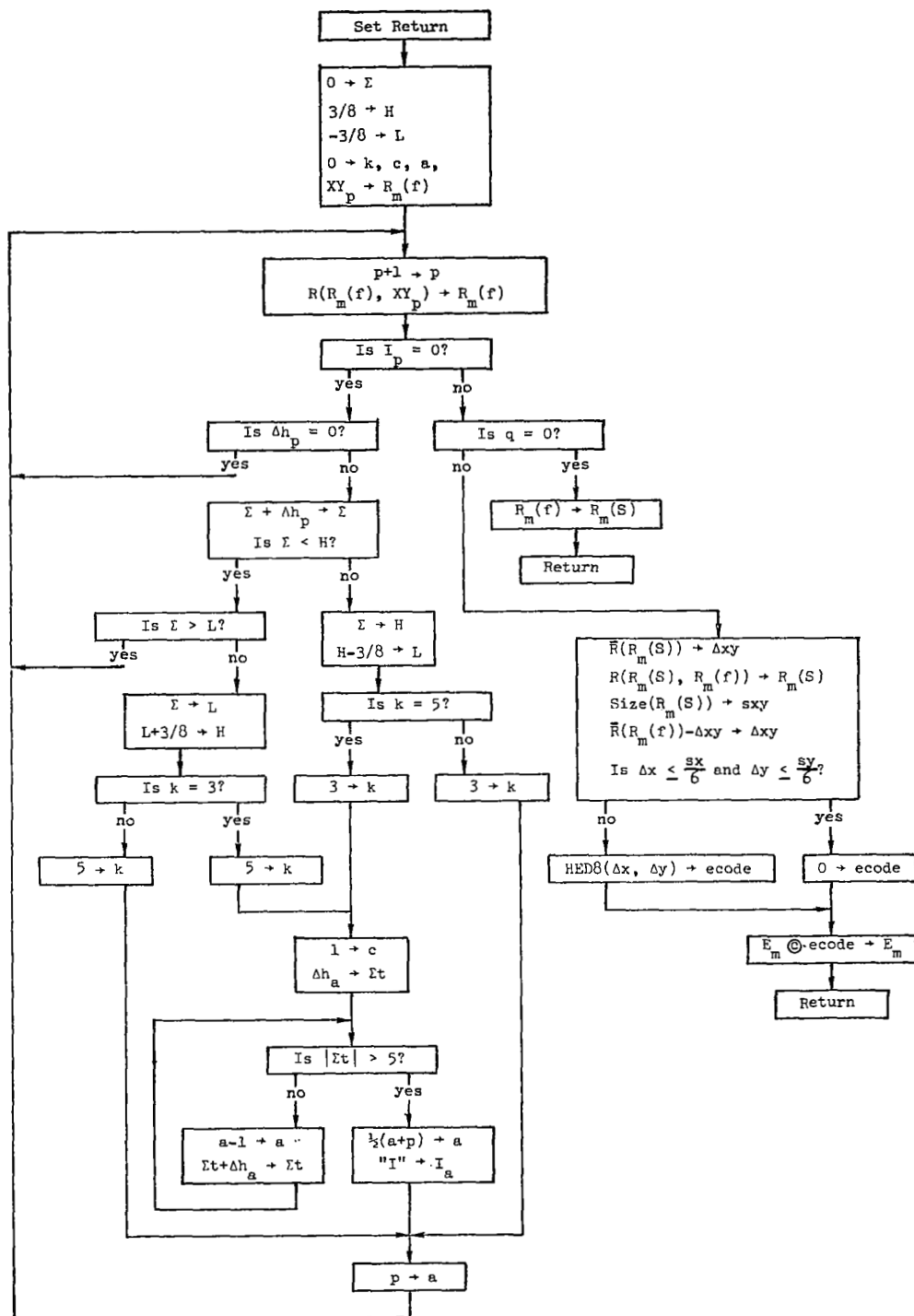


Figure 15. INFLEX Subroutine

rotating stroke. If neither threshold is exceeded, the segment or stroke is treated as a straight line by BOTH. As soon as one of the thresholds has been exceeded, the other is modified so that the difference is $3/8$. Only after one and then the other of the two thresholds has been exceeded (using the proper signs) is it assumed that an inflection point exists. Then a search is made backward from the point that contributed the amount of curvature needed to exceed the second threshold to the best approximation of the inflection point. This determination is made by having the index "a" pointing to the last point which exceeded a limit the first time, using "k" as a switch. A temporary sum is generated backwards from the point indexed by "a", decrementing "a" each time, until a limit of $5/16$ is exceeded. The inflection point is then marked half way between the point indexed by the last point examined (indexed by p) and the present point indexed by a. The detection process continues, marking each inflection found until either a corner or the end of the stroke is found.

6.8 XØVER SUBROUTINE

XØVER is entered from BØTH (see Figure 16) only if no inflection point is found and the total curvature found by INFLEX up to the present point exceeds 1 (that is, a half circle). The flag "k" is set by INFLEX to indicate the direction of curvature; 3 for clockwise and 5 for counter-clockwise. If the end-points of the segment or stroke are within 8 raster units of one another, or they are within one third of the width of $R_m(f)$ in x and one third the heights of $R_m(f)$ in y, the path is considered closed and the entire figure is taken as a loop. The code value stored by INFLEX with a zero added is used as the output feature code. Otherwise, the path is followed through the four quadrants (see Figure 13) until a quadrant is re-entered by the path and ended in or has exited the quadrant.

When this circumstance arises, MINPTS is entered to determine the two points on the path within the same quadrant that are closest to one another. MINPTS sets the appropriate feature code in CI_c for the condition found before returning to XØVER; it also indicates whether or not all of the stroke or segment has been examined in the process (xtry=2) so that XØVER can complete the feature string for the path. If no loop is found, XØVER generates a feature string based on the four quadrants of the minimum surrounding rectangle. Note that the center diamond is not used.

6.9 MINPTS SUBROUTINE

MINPTS (see Figure 17) is entered from XØVER with the indices q and p set to the beginning and end of the portion of the stroke that has begun and ended in the same quadrant of the minimum rectangle. If the path has not ended in the quadrant, a search is made forward (toward the end) to determine if there

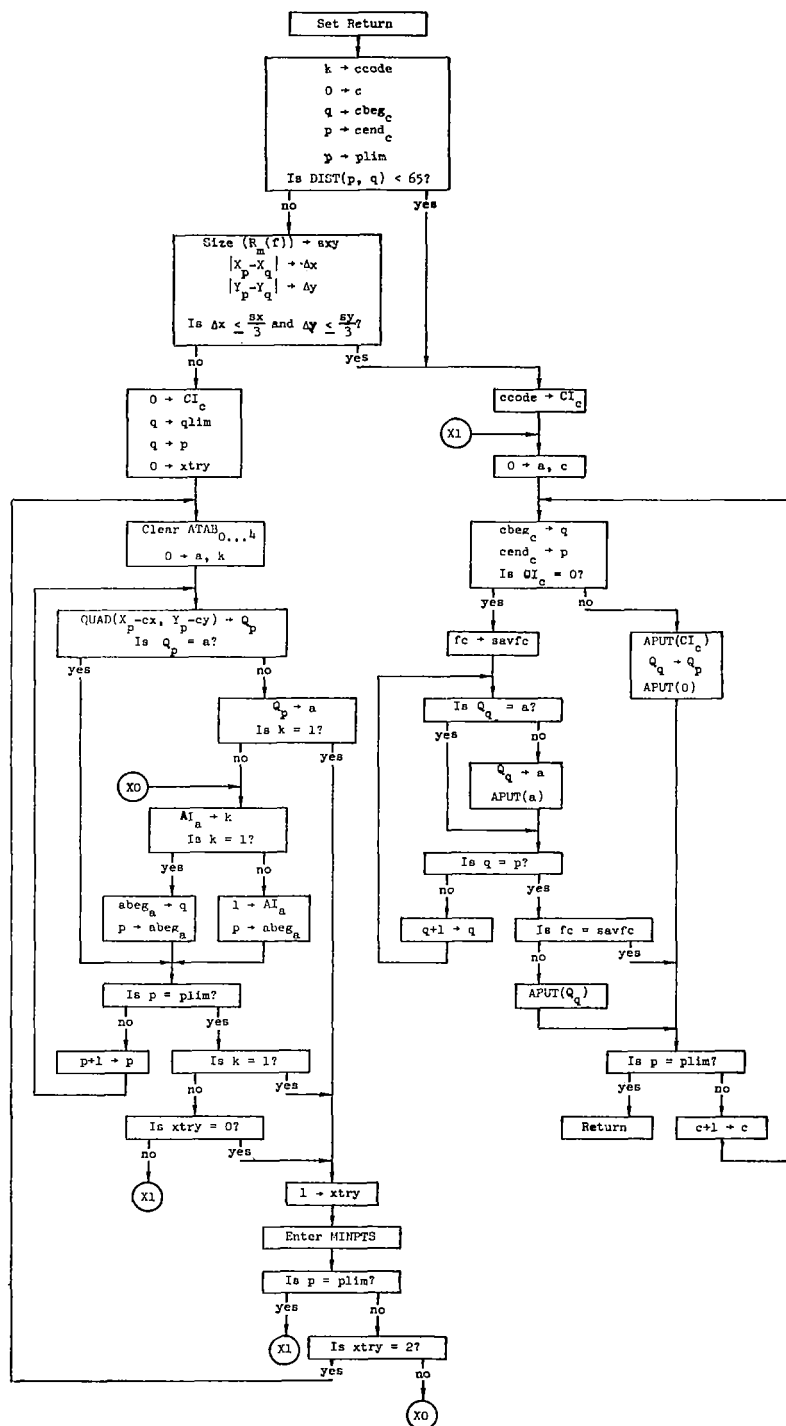


Figure 16. XOVER Subroutine

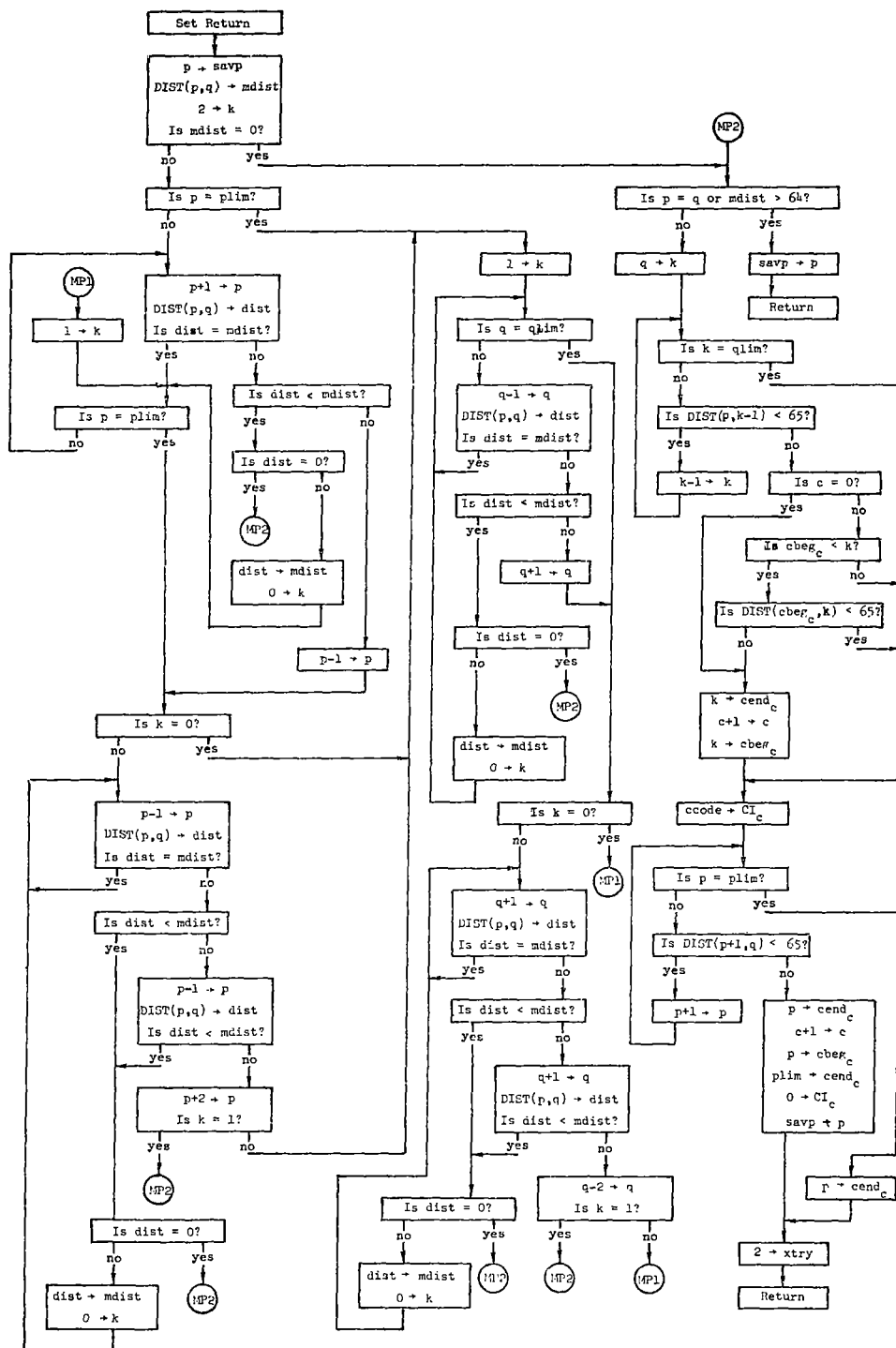


Figure 17. MINPTS Subroutine

is a pair of points (xy_{p+n} , xy_q) closer than the beginning and ending pair (xy_p , xy_q). By appropriately manipulating p and q , the pair of points that are a minimum distance from one another is determined. If that distance is less than or equal to 8 raster units (65 is used in MINPB because the routine DIST generates the square of the distance), a closed loop has been found; otherwise the path is not considered to have an intersection or to be closed, the appropriate quantities and tables are updated and control is returned to XØVER.

6.10 QUAD, APUT and DIST

These three subroutines (see Figure 18) are service routines used by BØTH, XØVER and MINPTS.

QUAD simply assigns a quadrant number based upon the Δx and Δy supplied by the calling routine. In all cases of its use, the calling routine supplies $\Delta xy = x, y - \bar{R}(f)$.

APUT simply concatenates the feature supplied when the routine is entered to the right end of the feature string (Fm), and increments the feature count (fc) by one.

DIST computes the square of the distance between two points xy_i and xy_j , i and j (indices pointing to two entries in the PTS table) supplied by the calling routine.

6.11 DEFINE SUBROUTINE

DEFINE (see Figure 19) adds definitions to the dictionary. The DEFINE routine starts with the last stroke of the input (entry m of the STRK table), inserting only those strokes that were not found by SEARCHD. If all of the required stroke information for the new character cannot be added because of insufficient space, none is added to the dictionary, thus eliminating the problem of dangling entries. In addition, it sets the variable "defn" to "OK" (if successful), or to "NG" if the above problem arises.

6.12 SEARCHD SUBROUTINE

SEARCHD (see Figure 20) determines from the index m if the stroke to be searched is the first stroke of an input character or a successor stroke. If the stroke is a first stroke, the routine SEARCHF is entered. Otherwise, SEARCHD computes the geometric relationship between the current stroke and the collection

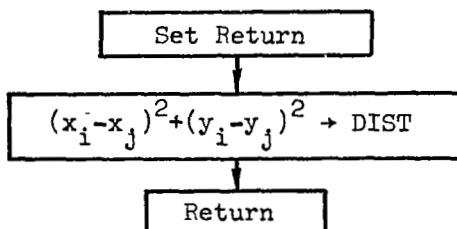
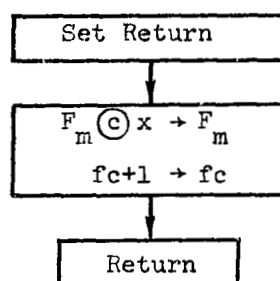
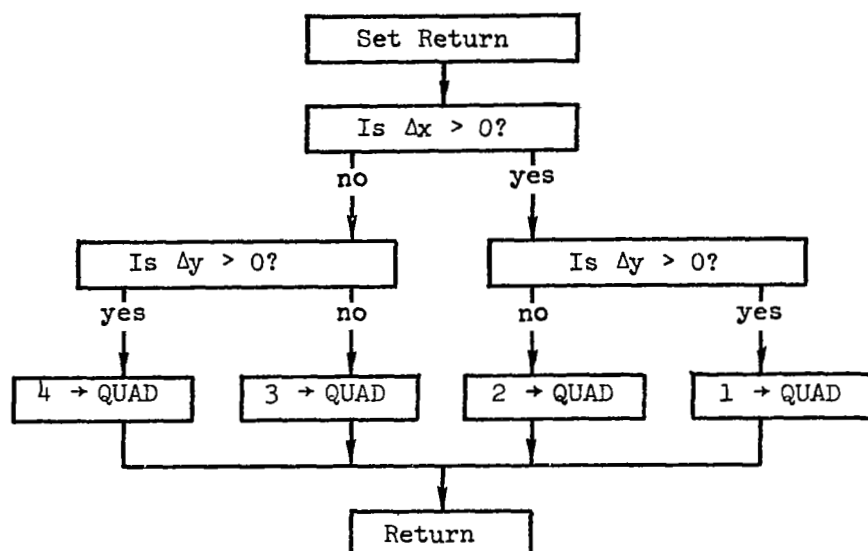


Figure 18. QUAD($\Delta x, \Delta y$), APUT(x), and DIST(i, j) Subroutines

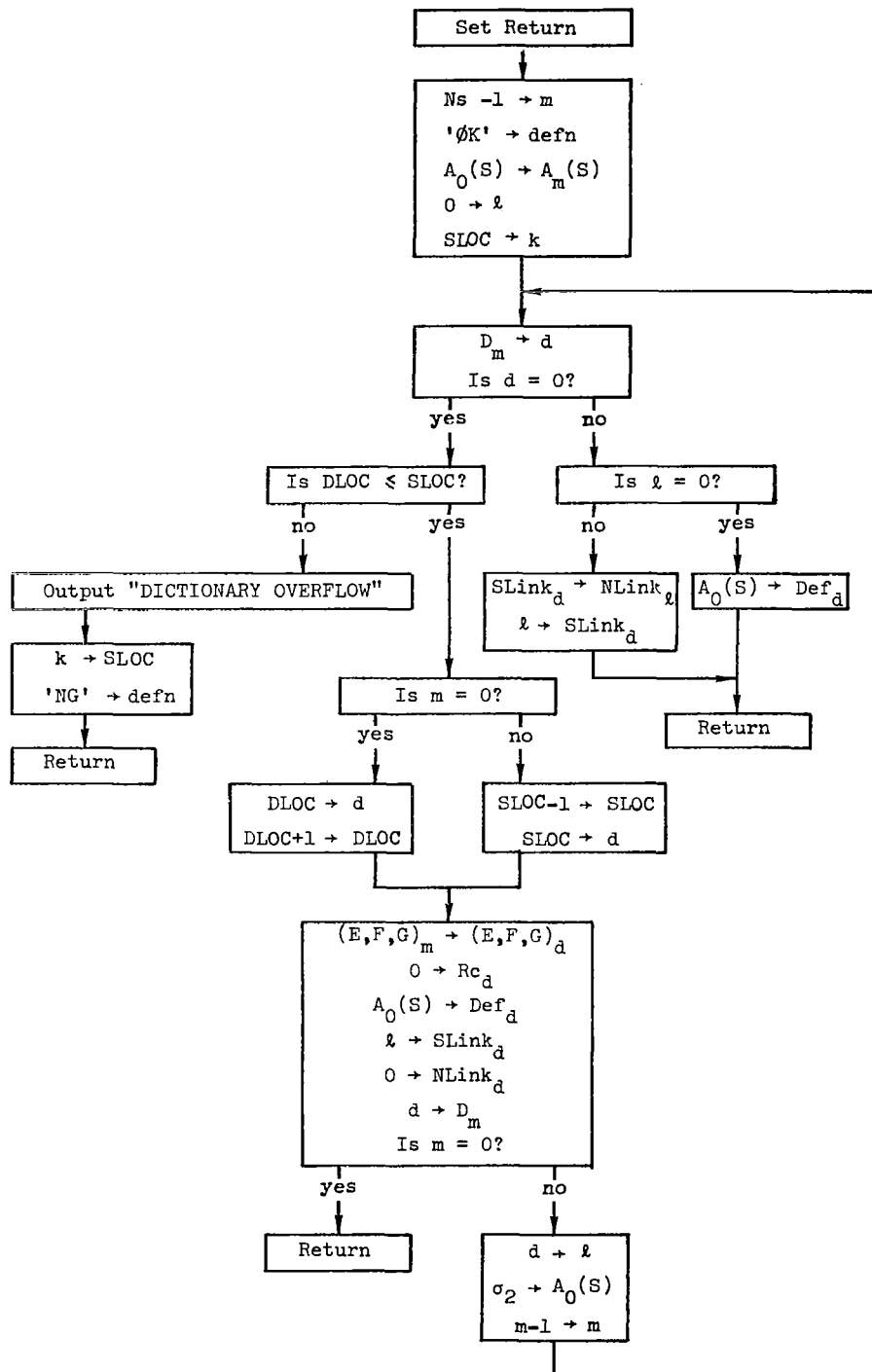


Figure 19. DEFINE Subroutine

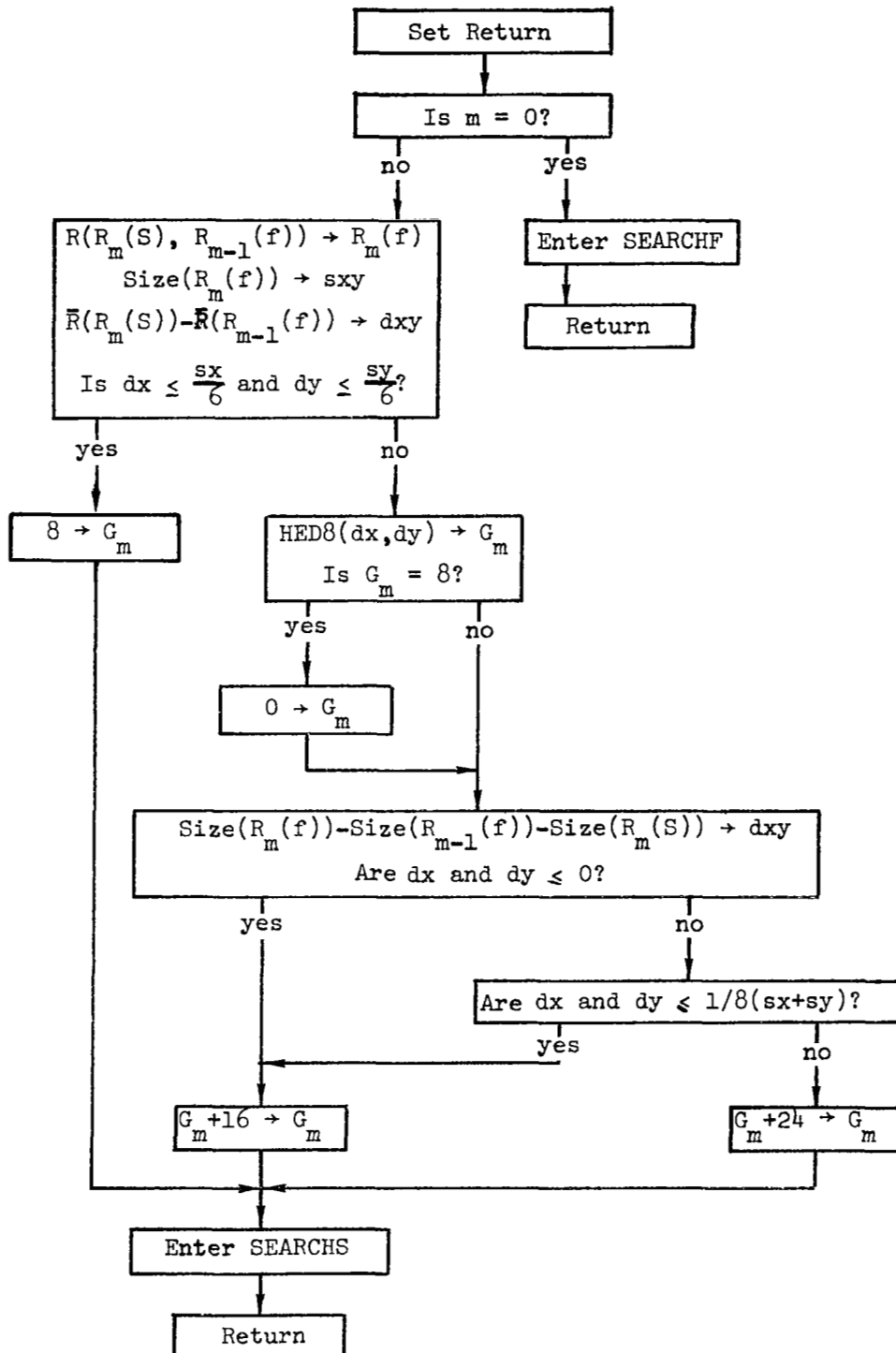


Figure 20. SEARCHD Subroutine

its predecessors. It first computes the minimum rectangle surrounding all strokes to this point and its size. It then differences the center coordinate of the rectangle surrounding the current stroke with that of its predecessors and determines if they are coincident. If they are, SEARCHS is called directly. (G_m is preset to zero and that is the code used for coincidence.) If the two rectangles are not coincident, the center-to-center direction is computed using the routine HED8, and inclusion or overlap and nearness are tested by computing

$$dxy = \text{Size}(R_m(f)) - \text{Size}(R_{m-1}(f)) - \text{Size}(R_m(S))$$

Figure 21 illustrates several examples of this computation. If from this test it is determined that the rectangles are "near," 16 is added to the center-to-center heading to complete the code for the geometric relationship; otherwise 24 is added to indicate that the rectangles are "far" from one another.

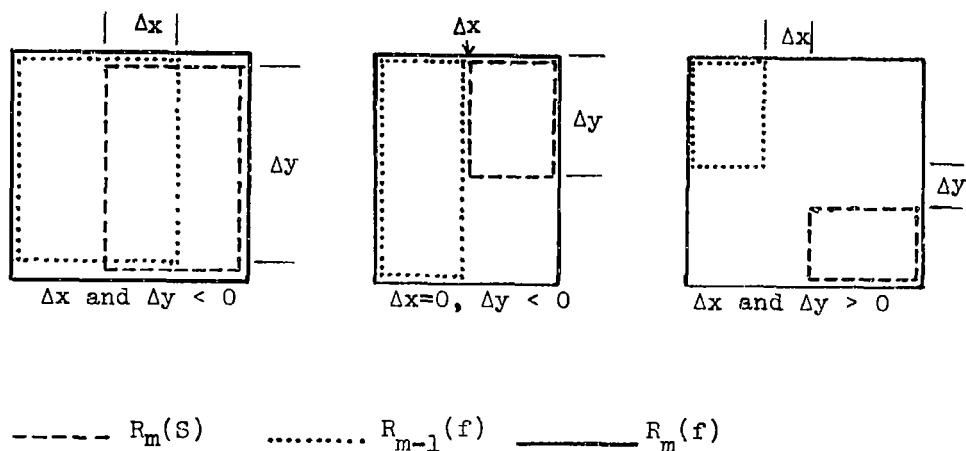


Figure 21. Examples of Overlap Computation

6.13 SEARCHF SUBROUTINE

SEARCHF (see Figure 22) begins its search for an exact match with the last entry of the first strokes in DICT. (Figure 23 shows the content and layout of DICT, the character definition dictionary.) If an exact match is found for the feature string of the unknown stroke, the routine exits with the dictionary location in D_m and the definition (Def) of that entry in $A_m(S)$.

If no match is made, D_m is set to zero and $A_m(S)$ is set to σ_1 .

6.14 SEARCHS SUBROUTINE

SEARCHS (see Figure 24) begins its search by examining D_{m-1} to determine whether or not a legitimate successor stroke exists in the dictionary. If D_{m-1} is not equal to zero, the successor link of the previous stroke must also be non-zero, otherwise the routine returns with D_m equal to zero and $A_m(S)$ set to σ_1 .

If a successor stroke exists, an exact match must be found between the feature string and geometric relationship of the unknown stroke, and the dictionary entry. All of the potential successor strokes are tested in this way until either an exact match is found or a "next link" equal to zero is found. In the former case, the routine returns with D_m equal to the definition located at that entry. In the latter case, the routine returns with D_m equal to zero and $A_m(S)$ equal to σ_1 .

6.15 HED8 and DIRQ SUBROUTINES

HED8 (see Figure 25) and DIRQ (see Figure 26) are two subroutines used for quantizing direction. Both routines work from tables. Their basic difference is the degree of fineness of the quantization. HED8 quantifies directions into one of 8; DIRQ into one of 32. The values shown in HED8 (Figure 25) are the values used. In order to facilitate differencing of the headings computed by DIRQ (item h of the PTS table) to form Δh , the output values are formed as two's complement fractions and are differenced as full word two's complement fractions. A difference of -1 always results from a 180° change of direction between the two headings; +1 never results, thus generating consistent values.

6.16 PURGE, MERGE and OPTIMIZE SUBROUTINES

These three programs do not run under the same control as those discussed above. Because they require communication that would be difficult through the Graphic Tablet Display console, they communicate with the user

DICT	E,F	-	Rc	Def	-	SLink	
+1	"	-	"	"	-	"	First Strokes
+2	"	-	"	"	-	"	
+3	"	-	"	"	-	"	
+4							
+5							Successor Strokes
+6							
+n-6							Successor Strokes
+n-5							
+n-4							
+n-3							
+n-2	E,F	G	Rc	Def	NLink	SLink	
+n-1	"	"	"	"	"	"	
+n	"	"	"	"	"	"	

← DLOC

← SLOC

Figure 23. Character Definition Dictionary

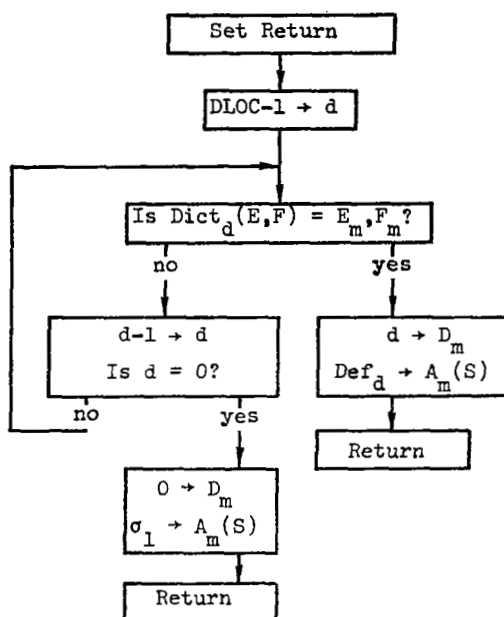


Figure 22. SEARCHF Subroutine

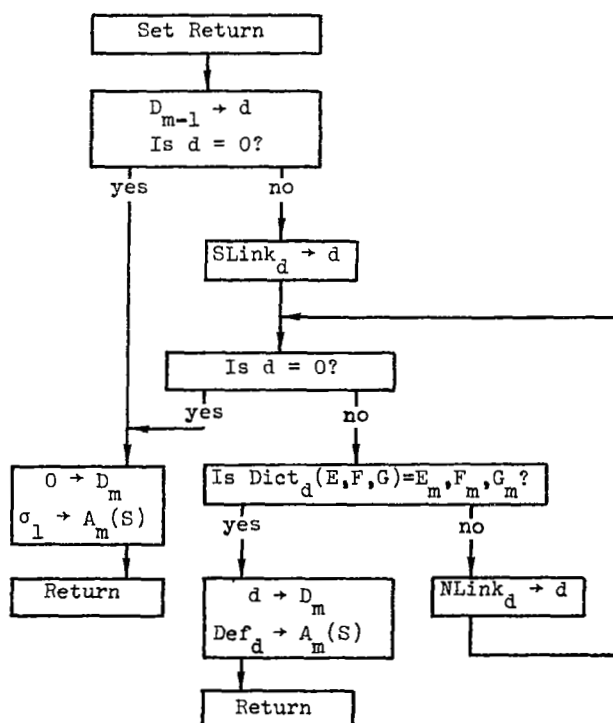
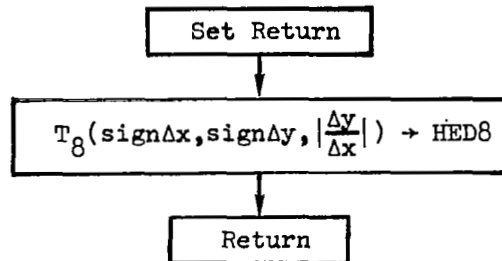
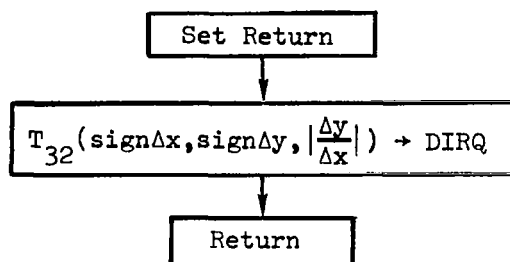


Figure 24. SEARCHS Subroutine



T_8	$\Delta x > 0$ $\Delta y > 0$	$\Delta x < 0$ $\Delta y > 0$	$\Delta x > 0$ $\Delta y < 0$	$\Delta x < 0$ $\Delta y < 0$
$\infty > \frac{\Delta y}{\Delta x} \geq \sqrt{2}+1$	8	8	4	4
$\sqrt{2}+1 > \frac{\Delta y}{\Delta x} \geq \sqrt{2}-1$	1	7	3	5
$\sqrt{2}-1 > \frac{\Delta y}{\Delta x} \geq 0$	2	6	2	6

Figure 25. HED8 (Δxy) Subroutine Flow Chart and Table



T_{32}	$\Delta x > 0$ $\Delta y > 0$	$\Delta x \leq 0$ $\Delta y > 0$	$\Delta x > 0$ $\Delta y \leq 0$	$\Delta x \leq 0$ $\Delta y \leq 0$
$\infty > \frac{\Delta y}{\Delta x} > 10.175$	0	0	-1	-1
$10.175 > \frac{\Delta y}{\Delta x} \geq 3.287$	$\frac{1}{16}$	$\frac{-1}{16}$	$\frac{15}{16}$	$\frac{-15}{16}$
$3.287 > \frac{\Delta y}{\Delta x} \geq 1.871$	$\frac{1}{8}$	$\frac{-1}{8}$	$\frac{7}{8}$	$\frac{-7}{8}$
$1.871 > \frac{\Delta y}{\Delta x} \geq 1.219$	$\frac{3}{16}$	$\frac{-3}{16}$	$\frac{13}{16}$	$\frac{-13}{16}$
$1.219 > \frac{\Delta y}{\Delta x} \geq 0.821$	$\frac{1}{4}$	$\frac{-1}{4}$	$\frac{3}{4}$	$\frac{-3}{4}$
$0.821 > \frac{\Delta y}{\Delta x} \geq 0.534$	$\frac{5}{16}$	$\frac{-5}{16}$	$\frac{11}{16}$	$\frac{-11}{16}$
$0.534 > \frac{\Delta y}{\Delta x} \geq 0.303$	$\frac{3}{8}$	$\frac{-3}{8}$	$\frac{5}{8}$	$\frac{-5}{8}$
$0.303 > \frac{\Delta y}{\Delta x} \geq 0.098$	$\frac{7}{16}$	$\frac{-7}{16}$	$\frac{9}{16}$	$\frac{-9}{16}$
$0.098 > \frac{\Delta y}{\Delta x} \geq 0.0$	$\frac{1}{2}$	$\frac{-1}{2}$	$\frac{1}{2}$	$\frac{-1}{2}$


Figure 26. DIRQ (Δxy) Subroutine Flow Chart and Table

via a keyboard console. This imposes no hardship, since the keyboard console is required to initiate loading and other communication required by the Time-Sharing System.

6.16.1 PURGE Subroutine

When called from the keyboard console, the program asks the user to supply a purging threshold--that is, the recognition count (Rc) level below which entries are to be removed. (It should be remembered that ANALYZER increments the recognition count of an entry each time a completely successful match is made between an input character and a dictionary entry.) If the user replies "NONE," the program asks which characters or entries, regardless of recognition count, are to be removed. This interactive conversation is not actually a part of the PURGE routine (see Figure 27), but rather is a part of an interactive keyboard control program that provides the interface between the user and the various service and special debugging aids that are a part of the system.

After the user supplies the appropriate response, the PURGE program methodically searches the dictionary, entry by entry, following the links comparing the recognition count to the user-supplied threshold--called Thresh--and output character code--called PChar (only one is valid). PURGE marks those entries that meet the criteria as "undefined" entries, and sets the recognition count for those entries to zero. It then removes and restructures the appropriate links for first-stroke entries. In order to accomplish this task, it uses an additional table, DICTE.

When it has examined every entry, PURGE then enters COMPACT (see below), a program that does the restructuring of the dictionary to recover space of the vacated entries. Note that in the case of multi-stroke characters, only those strokes of the definition are removed that are not linked to some other entry that is not to be removed. That is, only the last n strokes of an m-stroke definition may actually be physically removed from the dictionary. Take, for example, a four-stroke "M" drawn as  where the first three strokes are defined as an "N". If the user requests that "M's" be removed from the dictionary, only the fourth stroke of the "M" in question will actually be removed and the first three strokes that define an "N" will be undisturbed.

6.16.2 COMPACT Subroutine

COMPACT (see Figure 28) starts with the first of the successor strokes, compacting the entries by removing those indicated in the DICTE tables by the calling program, and restructuring the links appropriately so that the entries are tightly packed. It then restructures the links in the first stroke and the successor stroke entries from the information saved in the DICTE table.

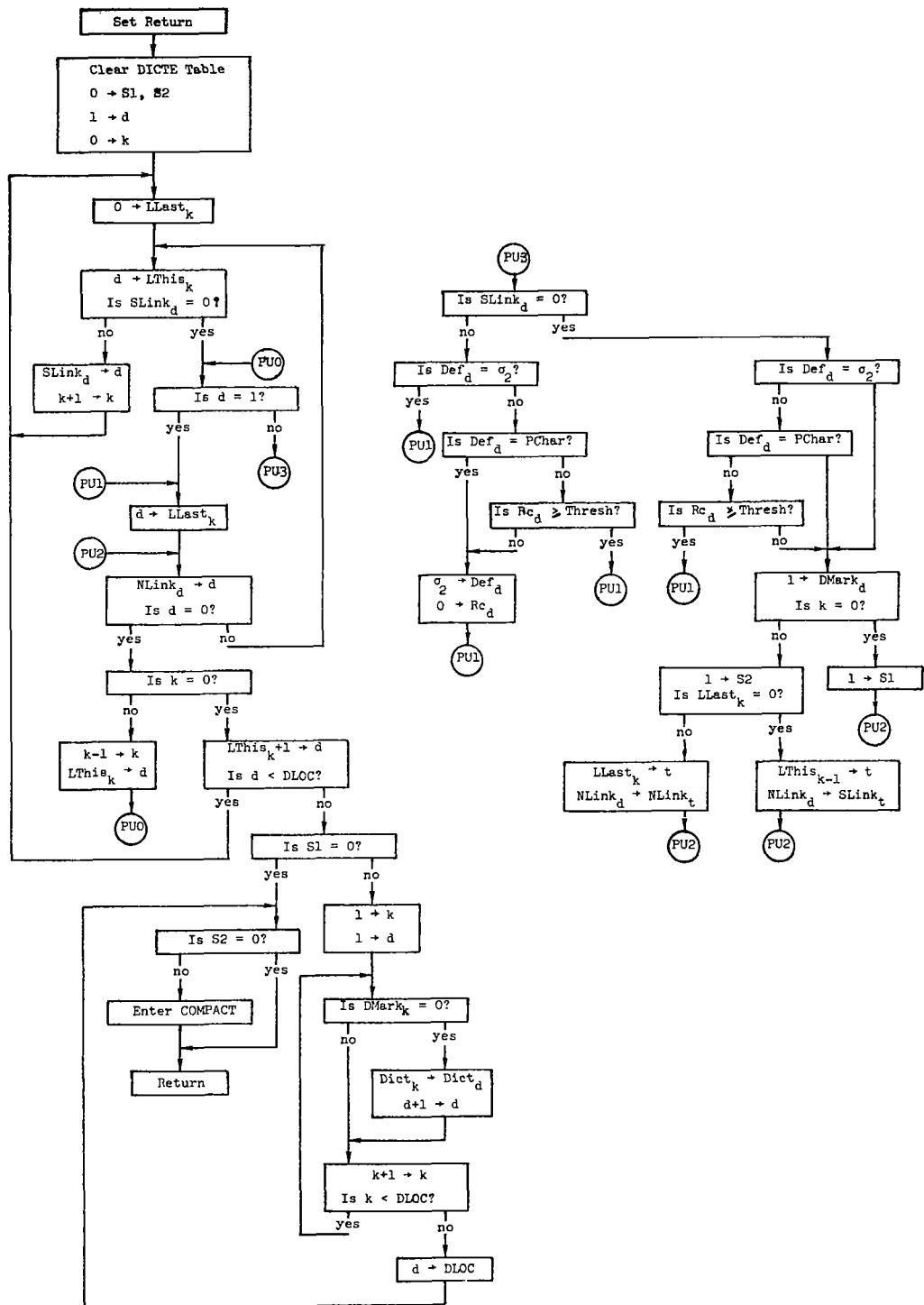


Figure 27. PURGE Subroutine

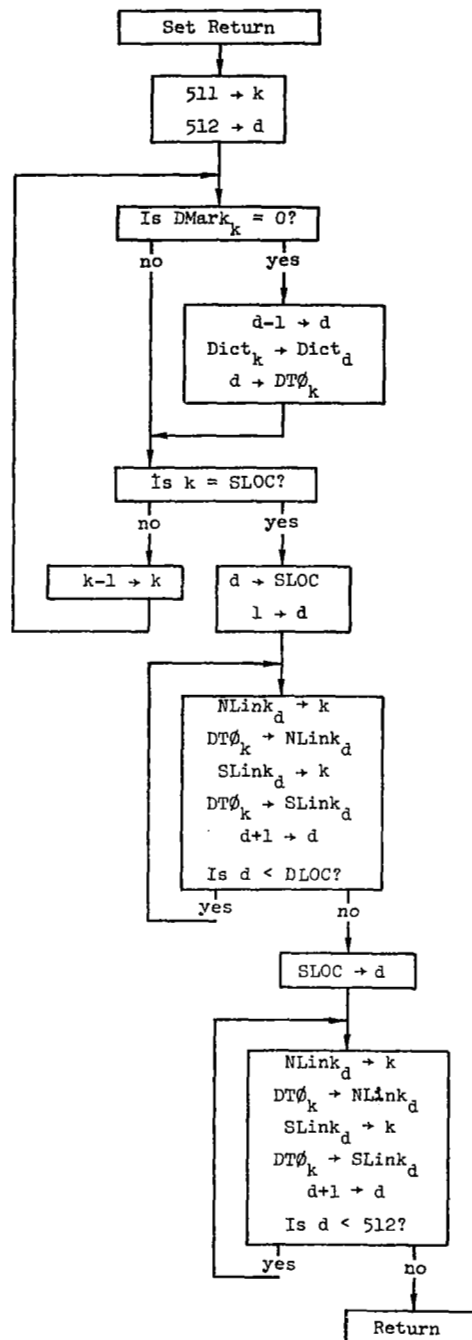


Figure 28. COMPACT Subroutine

6.16.3 MERGE Subroutine

The interactive keyboard control program requires the user to specify the file name of the dictionary that is to be merged into the existing one. When it receives a proper name from the user ("proper" in this sense means that it is the name of a file that the user may access), it reads the dictionary file from the system's disc storage into a second dictionary space called DICT*. MERGE (see Figure 29) then takes the strokes from DICT* following the links, as if they were coming from the SAMPLE program, building a STRK table and searching the existing dictionary for matches. If a complete match is found, the sum of the recognition counts plus one replaces the recognition count in DICT. If no match or an incomplete match is made in DICT, the non-existent strokes are added to DICT by calling DEFINE.

In the case where there is an exact duplicate of feature information but the definitions in the two dictionaries differ, the user is asked to make a choice between the two available characters or delete the entry entirely. If he chooses the latter, it is noted and PURGE is called to delete the troublesome definitions. If the result of the merge produces a dictionary that is too large, the user is informed that the merge was incomplete, and he is unfortunately left with a dictionary of unknown content. Thus, it pays to have previously saved a copy of both dictionaries that are to be merged and to have purged and optimized at least one (if not both) dictionaries before beginning the merge. Our present system permits the user to save any dictionary on the system disc storage (at least temporarily) with any arbitrary name attached. Therefore, it is not unreasonable to keep several versions of the same dictionary available in case of emergency or for experimentation.

6.16.4 Optimization

The process of optimizing the dictionary is one of recognizing groups of dictionary entries that are unambiguously equivalent, and re-linking the dictionary to reflect this equivalence.

Two dictionary entries are considered to be equivalent if they are on the same dictionary level (i.e., both first strokes, both second strokes, etc.), and one or more of the following conditions also holds true:

1. They have the same character definition.
2. They have successors at a common dictionary level that have the same character definition.
3. They are identical (in feature, envelope, geometric-relation) and their most immediate predecessor strokes are equivalent.

In forming equivalence groups, an additional rule is applied: two entries,

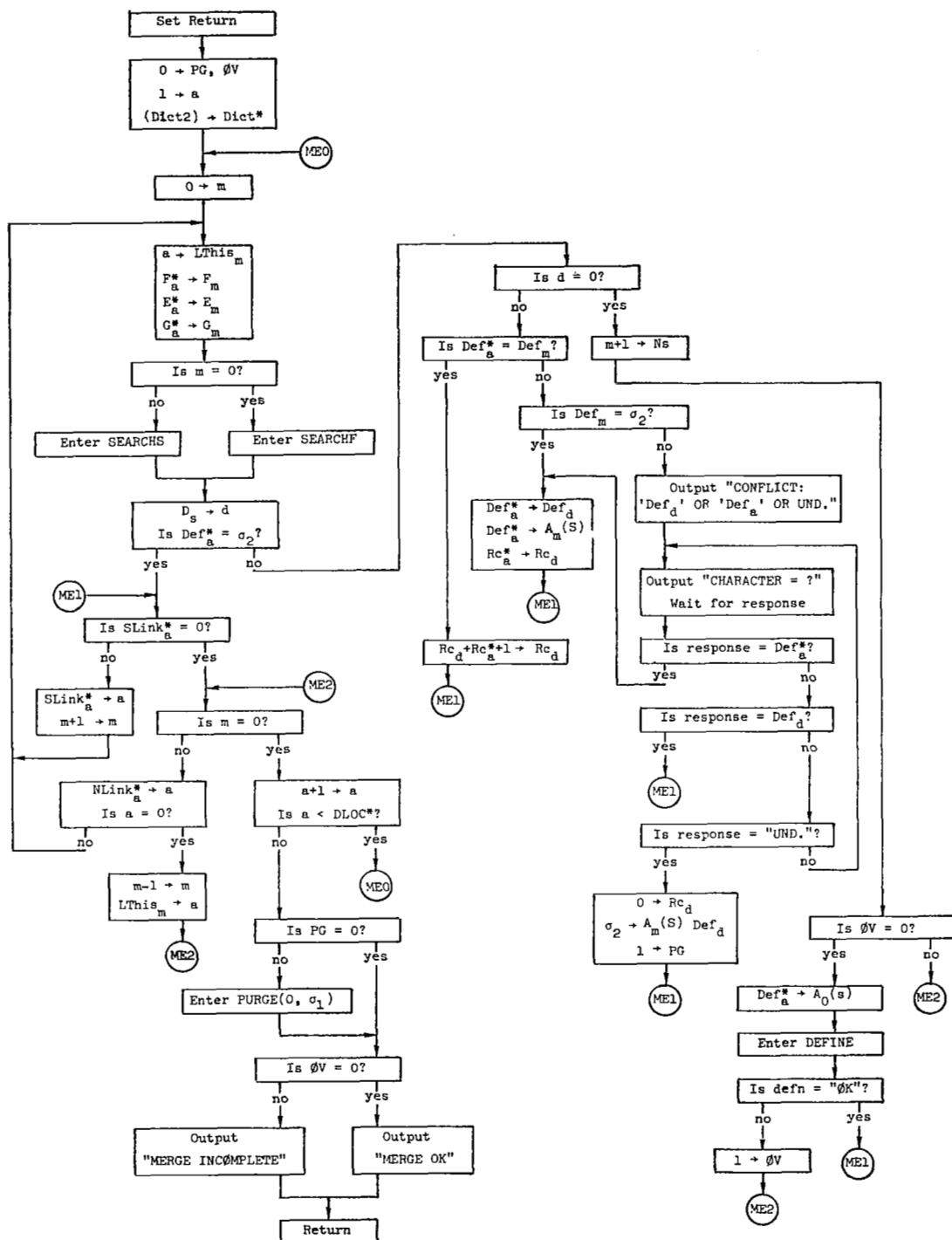


Figure 29. MERGE Subroutine

each equivalent to the same third entry, are also equivalent to each other. Finally, a dictionary entry that is equivalent to no other dictionary entries forms a one-member equivalence group by itself.

An ambiguous equivalence group is one in which members have conflicting character definitions; that is, among those members of a group which are not intermediate strokes, more than one unique character definition is present.

The optimizer divides the dictionary first-stroke section into a set of equivalence groups. Then, taking each of these groups separately, it further groups all successors, at all levels, to members of this group. The grouping process results in a tree structure whose nodes are equivalence groups. Each group, except the first, is a next-level successor group to some previous group. For each tree structure, ambiguity at any node renders the whole tree ambiguous, and the dictionary entries associated with the tree are not optimized. Otherwise, three kinds of optimization are performed:

1. Where a group contains subgroups of identical dictionary entries, all but one member of each subgroup are deleted from the dictionary.
2. When a group contains members that are intermediate strokes as well as members that have a character definition, the intermediate strokes are redefined to that character.
3. All members of each successor group are linked, in the dictionary, as next-level successors to all members of its parent group. This provides definition paths in the dictionary which did not before exist.

6.16.5 OPTIMIZE Subroutine

Optimization is performed by procedure OPTIMIZE (see Figure 30) and the procedures DTREE, ADDGROUP, GTREE, GCHECK, RELINK, and COMPACT which it calls. OPTIMIZE determines the membership of an equivalence group at any level from information contained in the section of the SORT table for that level; thus, before any grouping is performed, the first-level section of the SORT table is constructed by applying procedure DTREE to each first-stroke entry of the dictionary.

6.16.5.1 DTREE Subroutine

When procedure DTREE (see Figure 31) is applied to a dictionary entry (entry_p), it examines entry_p and its successors at all lower levels. For each such entry that has a character definition (is not an intermediate stroke), it

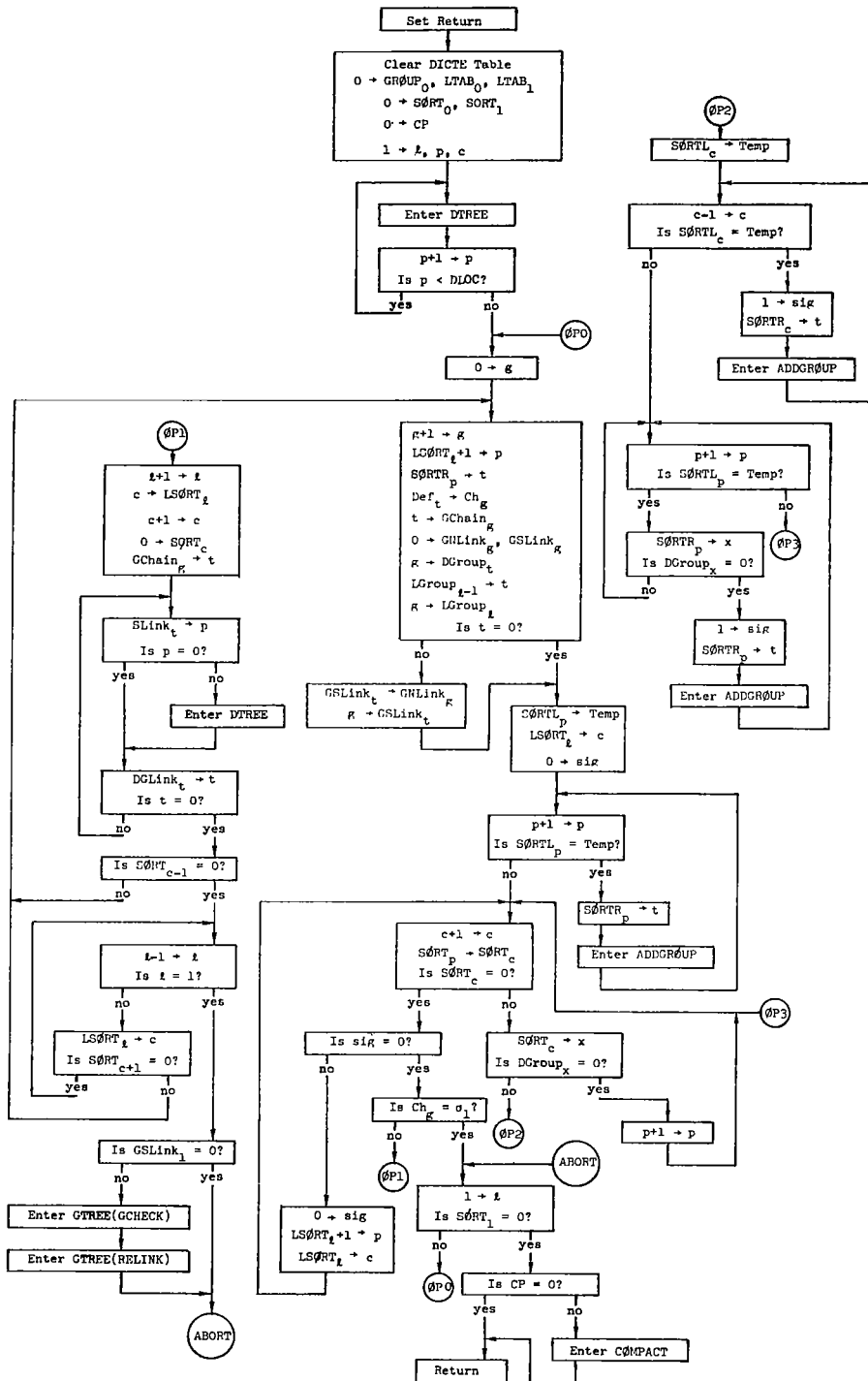


Figure 30. OPTIMIZE Subroutine

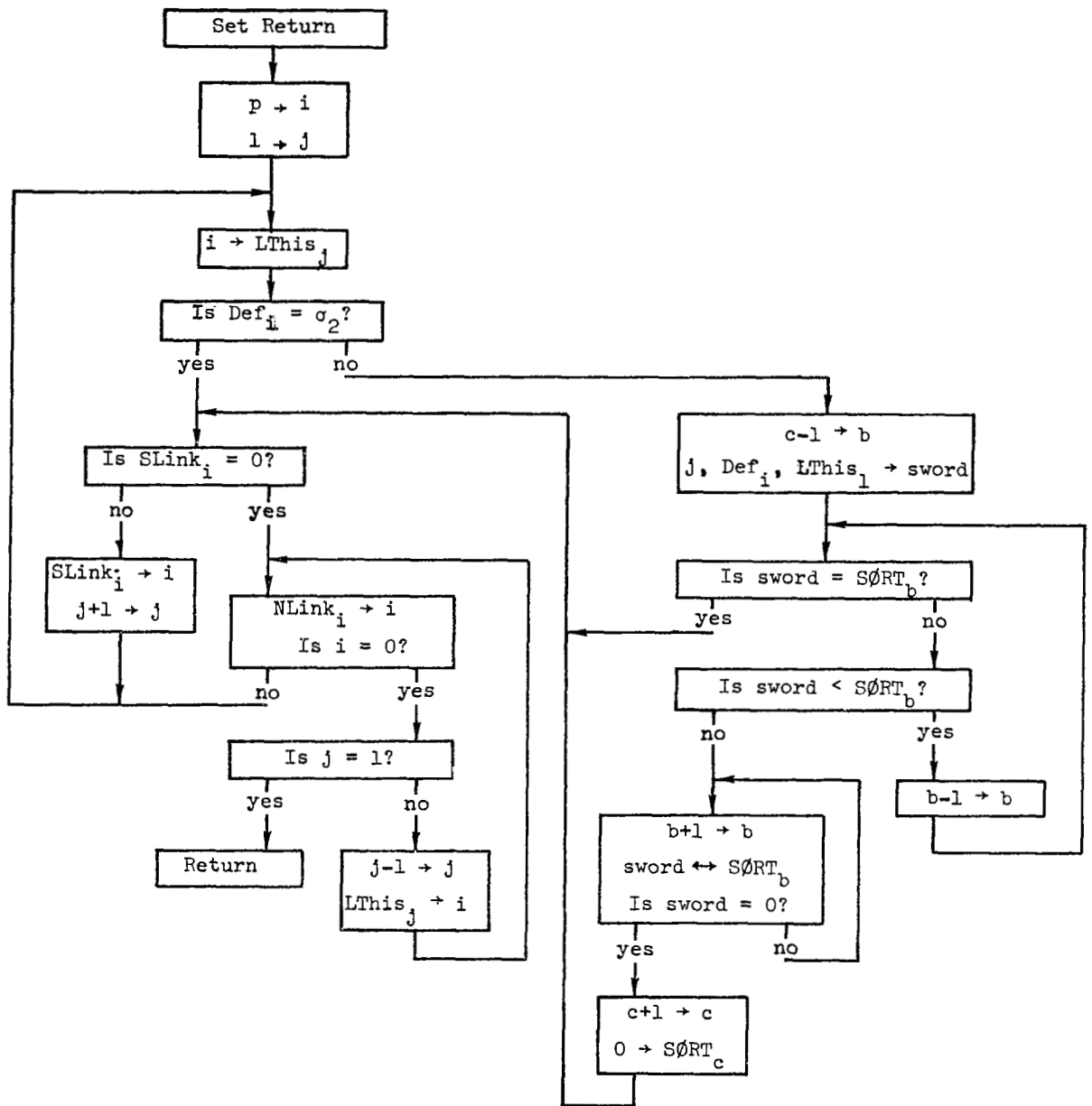


Figure 31. DTREE Subroutine

produces a tentative SORT table entry, which is then added to the current section of the SORT table, only if no identical entry already exists there. SORT entries are added so as to maintain ascending sort order within each section, and section brackets (zero-words) are maintained at the top and bottom of each section. (The bottom bracket of each section also serves as the top bracket of the next section.)

A SORT entry is in two pieces:

1. SORTL, the left half-word, contains level (relative to the level of entry_p, which is assigned level 1), followed by the character definition.
2. SORTR, the right half-word, contains p.

Thus, when applied to entry_p, DTREE adds SORT entries for each unique dictionary character of which entry_p is a component stroke. The uniqueness test distinguishes, for example, between a 2-stroke and a 3-stroke "N".

Once the first-level SORT section has been built, OPTIMIZE attempts to optimize successor "families" of dictionary entries. For each family it first abstracts a first-level equivalence group from the SORT table, and then goes on to group the dictionary successors, at all levels, of members of the group.

The process to form a group, at any level, is this: The first group member is that referenced in SORTR of the first entry of the appropriate SORT table section. Additional members are added to the group where they are found to be paired (in SORTR) with a SORTL that is also paired with an existing group member. As iterations are made through the section, SORT entries that reference members of the group are deleted in such a way that when the search is exhausted, the remaining entries in the section (and the bottom bracket) have been compacted upwards.

As each group is formed, an entry in the GROUP table is constructed. GROUP entry₀ is never used; the first-level group goes into entry₁, and each successor group into the next available entry. Items GSLink and GNLink are used to link the various groups (nodes of the group tree). GSLink points to a group's first successor group; each group in the successor chain points to the next link in its GNLink. Item GChain is the head of the group's membership chain; it points to the dictionary location of the first group member. Each member is linked to the next member in item DGLink in the extended dictionary (table DICTE). Item DGroup (also in DICTE) points, for each member, to the GROUP table entry of its group. Item Ch contains the group's character definition.

When OPTIMIZE establishes a new group with its first member, it sets Ch to the dictionary definition (Def) of that member. (This may be an intermediate stroke, or σ_2 .)

6.16.5.2 ADDGROUP Subroutine

ADDGROUP (see Figure 32) adds a new member to the front of a group's membership chain. It also sets the new member's DGroup, and updates the group's Ch. If Ch starts as σ_2 , it is replaced by the definition of the first new member that has a Ch $^\emptyset$. Subsequent addition of a new member with a different character definition makes the group ambiguous. For ambiguous groups at other than the first level, an immediate abort is made. To ensure that all members of an ambiguous first-level group are removed from the SORT table, the group's Ch is set to σ_1 ; it remains so until the group is completely formed, at which time an abort is made.

An abort discontinues processing of the current family. The contents of the GROUP and SORT tables beyond the first sections are disregarded, and processing of the next family is initiated by forming a new first-level group into GROUP entry₁.

Upon formation of an unambiguous group, additional groups are formed in the following order:

1. The DTREE subroutine processes each member of the group just completed to form a SORT section at the next level. If this section is empty, control passes to step 2. Otherwise, the level is stepped down, and a group is formed from the new section to begin the successor chain of the group just made.
2. If the SORT section at the current level is not empty, another group is made at this level, and added to the front of the successor chain of the most recent group formed at the previous level. If the SORT section at this level is empty, however, the level is stepped back up, and step 2 is re-entered.

The process stops when step 2 is entered and level is 1. That is, the tree is completed when the next group to be made is a first-level group. To facilitate group linking and level stepping, table LTAB is maintained during the grouping process. Indexed by level, LTAB entries contain item LSORT (which points to the top bracket of the corresponding SORT section) and item LGroup (which points to the GROUP table entry for the most recently formed group at each level).

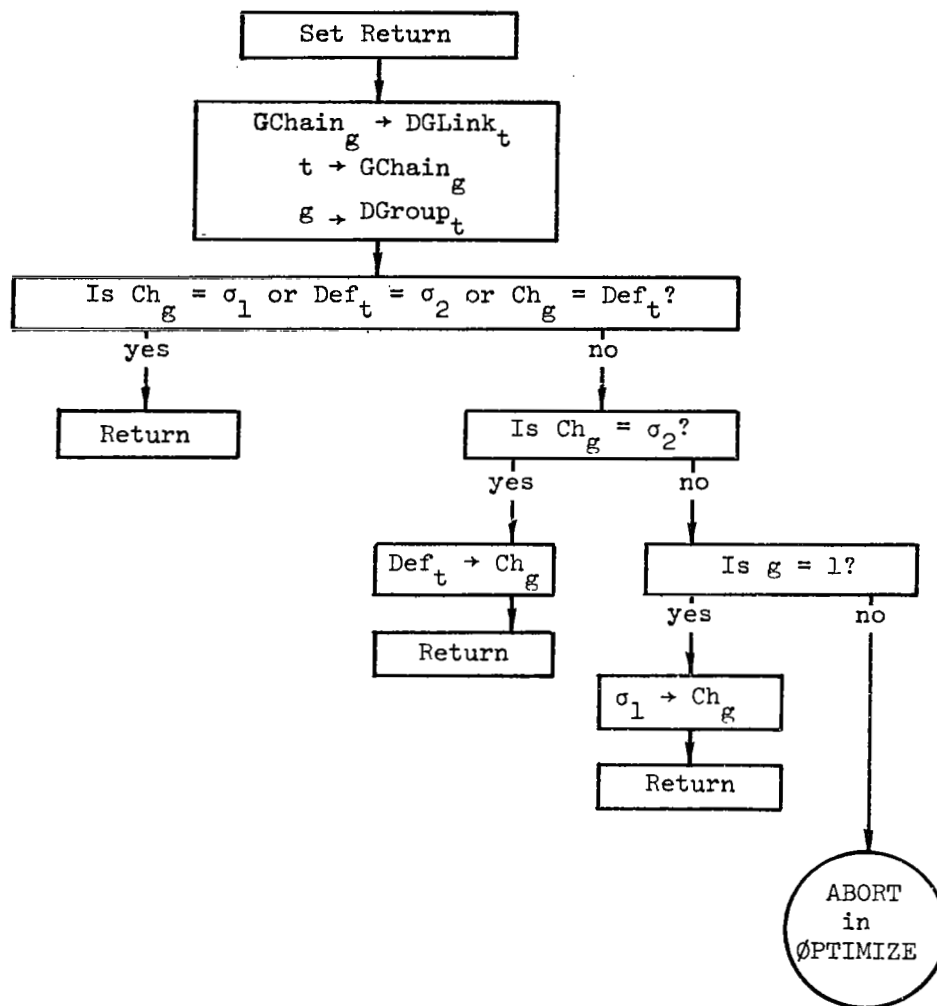


Figure 32. ADDGROUP Subroutine

A group tree which survives to this point has been constructed by the application of only the first two equivalence rules. To apply the identity rule, OPTIMIZE now calls procedure GTREE to step through the tree structure, applying procedure GCHECK to each group.

6.16.5.3 GTREE Subroutine

GTREE (see Figure 33) steps through the tree, beginning with the first-level group, in this order:

1. Down to the next level, to the group pointed to by GSLink, but if GSLink is zero, then:
2. Across to the next group pointed to by GNLink, but if GNLink is zero then:
3. Back up a level, then proceed to step 2.

GTREE is finished when all nodes have been processed. To facilitate stepping, GTREE uses the LEVEL table, in which item LThis points to the current node at each level.

6.16.5.4 GCHECK Subroutine

When GCHECK (see Figure 34) is applied to GROUP_g, it makes identity comparisons between each member of GROUP_g and all next-level dictionary successors to the members of GROUP_x, the parent group of GROUP_g. (No member of GROUP_g that has been marked as identical is used as a basis for comparison, however, and no member of GROUP_g is compared with itself.) When an identical dictionary entry is found, it is marked by having its DEQ in the extended dictionary point to the dictionary entry to which it is identical. If the identical entry is not also a member of GROUP_g, it belongs to another group (GROUP_i) which is also a successor to GROUP_x. In this case, GROUP_i is merged with GROUP_g. (But if the Ch's of these two groups are in conflict, the merged group would be ambiguous, and an abort to begin processing the next family is made.) Steps in the merging process are:

1. GROUP_i is deleted from the chain of successors to GROUP_x, and the linkage is closed (note that because of processing order, GROUP_i is further along this chain than GROUP_g, and has not yet been processed by procedure GCHECK).

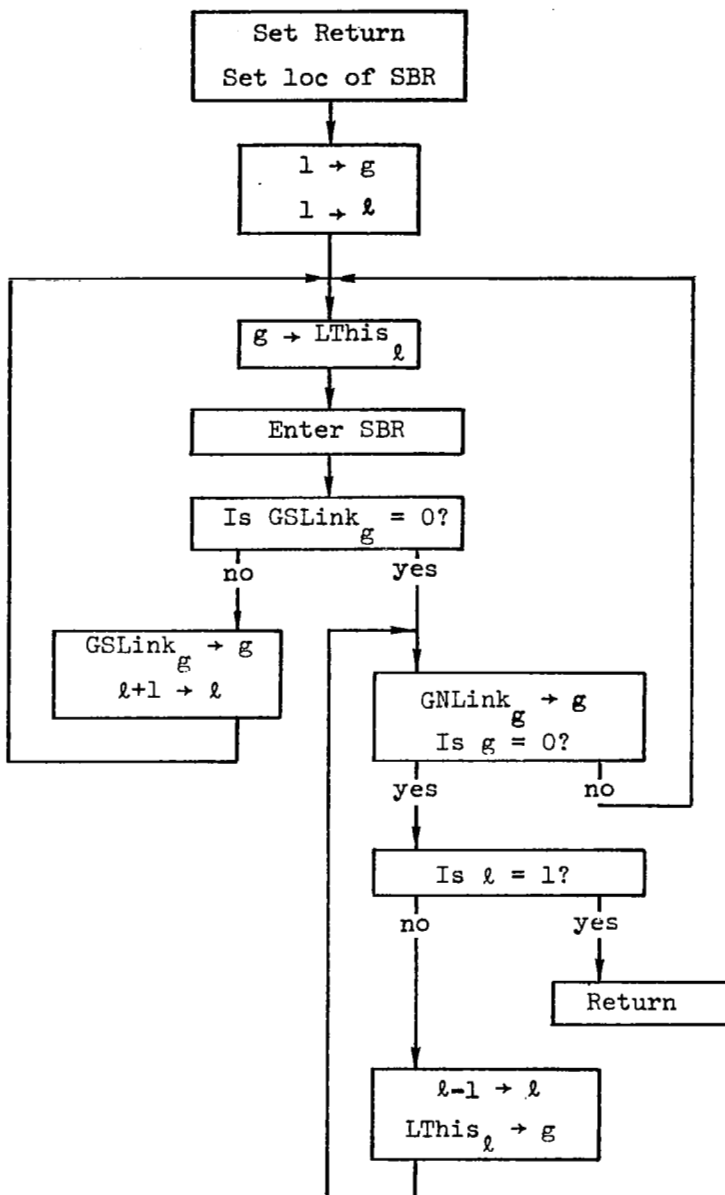


Figure 33. GTREE Subroutine

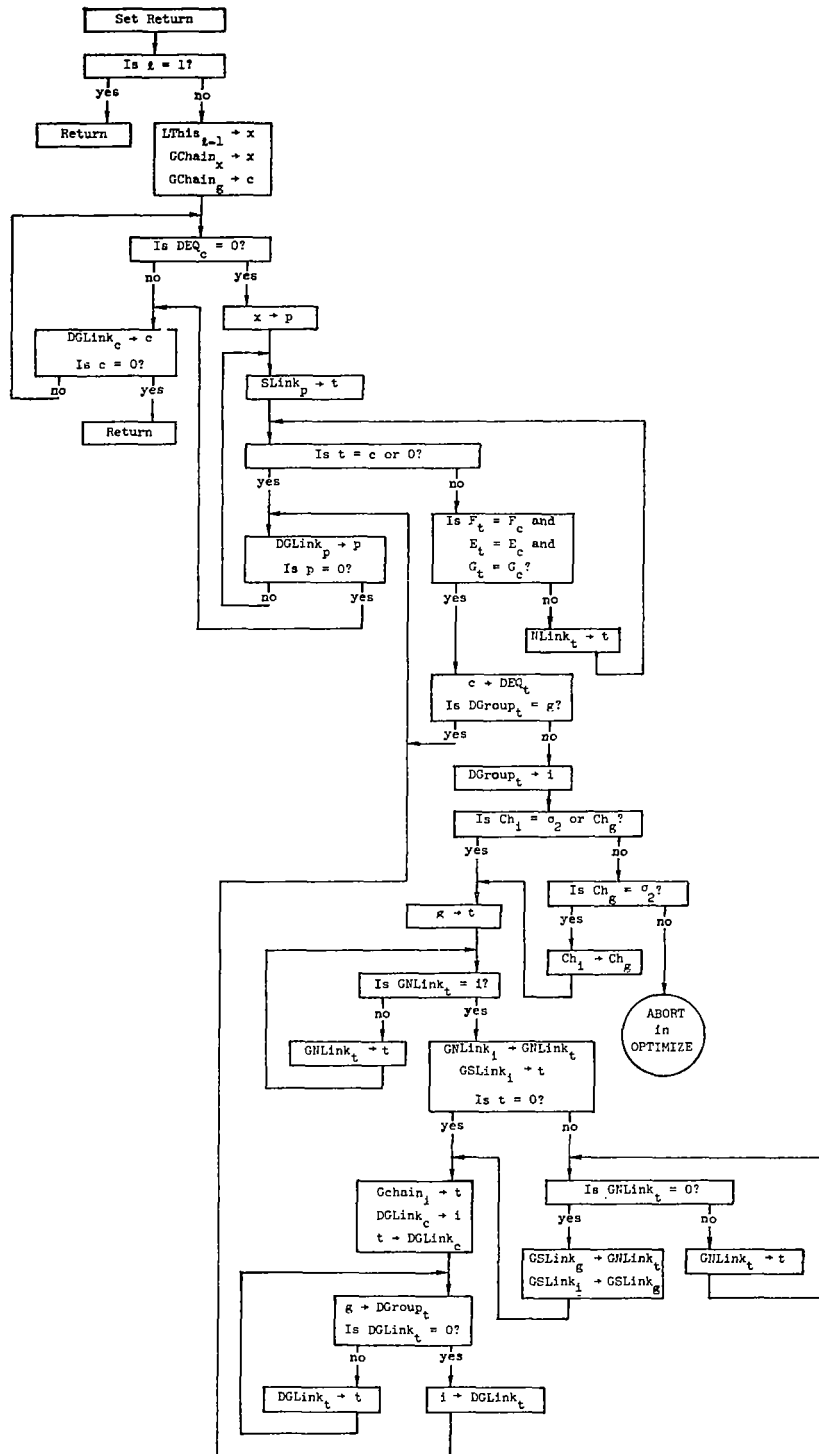


Figure 34. GCHECK Subroutine

2. If GROUP_i has a non-empty successor chain, then GROUP_g 's successor chain is appended to it, and the pointer GLink_g is made to point to the first link of the combined chain.
3. The chain of members of GROUP_i is inserted into the chain of members of GROUP_g immediately following the link which is currently the basis for comparison.
4. The DGroup of each member of GROUP_i is changed to reflect membership in GROUP_g .
5. If GROUP_g 's Ch is σ_2 , it is replaced by Ch_i .

Note that the GCHECK subroutine does not operate on the first-level group (GROUP_1); it has no parent group.

If a tree survives to this point, the dictionary entries associated with it are then optimized. OPTIMIZE again calls procedure GTREE to step through the tree, but this time applying procedure RELINK to each group.

6.16.5.5 RELINK Subroutine

RELINK (see Figure 35) applied to group g first considers the set of dictionary entries formed by the union of the members of all groups on GROUP_g 's successor chain. Each member of the set that has been marked as identical by GCHECK is processed in the following way:

1. It is eliminated from the set.
2. Its recognition count is added to the recognition count of the dictionary entry to which it is identical.
3. It is marked for subsequent deletion. Its DMark in the extended dictionary is set "on", and the deletion signal (item CP) is also set "on".

The remaining members of the set are linked together in the dictionary as the successor chain to each member of GROUP_g . SLink of each member of GROUP_g points to the head of the chain (SLink's are zero if the set is empty), and members of the chain are linked to each other by their NLinks.

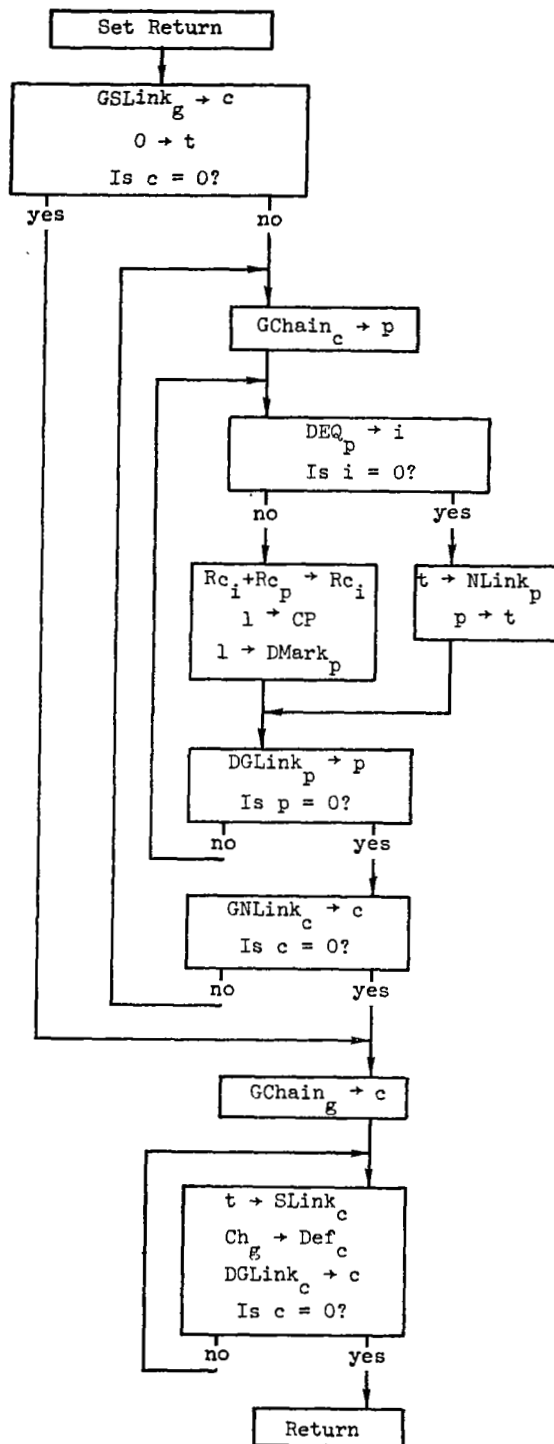


Figure 35. RELINK Subroutine

Finally, RELINK replaces the character definition of each member of GROUP with the group's Ch. Actual redefinition takes place when a member is an intermediate stroke and Ch is a defined character.

When relinking is finished, OPTIMIZE begins processing the next family. If at this time, or after an abort, the SORT table first-level section is empty, all families have been processed. Before exiting, if the deletion signal is on, OPTIMIZE calls the COMPACT subroutine (see above) to delete those dictionary entries that were marked by RELINK.

ADDENDUM A: GLOSSARY OF MNEMONICS AND ABBREVIATIONS

<u>TERM</u>	<u>MEANING</u>	<u>REFERENCE</u>
a, b, c, d, i, j, k, l, m, n, p, q, r, s	These letters are used to designate indices in the usual programming sense of pointers to table entries or iteration counts.	
A(S)	Result of processing an input stroke.	SAMPLE, TEST, ANALYZER STRØKE, BØTH, DEFINE, SEARCHF, SEARCHS
AI	A primary stroke feature-loop, inflection point, or corner.	XØVER
BINK	Origin of tablet input into the display buffer used to set INKØRG.	SAMPLE, TEST
C	Rejected point count.	GRID, STRØKE
ccode	Single feature storage.	XØVER
cbeg, cend	Beginning and end pointers.	XØVER, MINPTS
Ch	Definition associated with a group.	ADDGRØUP, GCHECK, RELINK
Ch ^Ø	Symbol used to designate a member of the output character set.	SAMPLE, TEST
CHSW	The program switch in SAMPLE that controls program flow and the meaning of user actions at the tablet. CHSW may be set to "D" meaning "a defined character has been output or a new definition added to the dictionary"; N, meaning "there is no	SAMPLE

<u>TERM</u>	<u>MEANING</u>	<u>REFERENCE</u>
	input" (this is the initial condition and the condition after a "clear"); "U" meaning "there is an undefined input pending; and "R", meaning "the function button REDEFINE has been pushed".	
CI	Indicator for direction of rotation.	XØVER
cix	Test value for index.	STRØKE
ct	Rejected point count.	ANALYZER
D	Dictionary pointer resulting from search.	ANALYZER, DEFINE, SEARCHF, SEARCHS
def	Output definition in each dictionary entry.	DEFINE, SEARCHF, SEARCHS, PURGE, MERGE, DTREE, ADDGRØUP, RELINK
defn	Indicator for success or failure of adding new entry to dictionary.	DEFINE, MERGE
DEQ	Pointer for like entries in dictionary.	GCHECK, RELINK
DGlink	Pointer into DICTE.	OPTIMIZE, ADDGRØUP, GCHECK, RELINK
DGroup	Classification for groupings.	OPTIMIZE, GCHECK
DI	Center diamond dimension.	BØTH
dist	Square of the distance between two points.	MINPTS
DLIST	The linked list of output characters in TEST containing size and position information and the relative location of the actual character in IMB or DB.	TEST

<u>TERM</u>	<u>MEANING</u>	<u>REFERENCE</u>
DLOØ	Pointer to next available first stroke location in dictionary.	DEFINE, SEARCHF, PURGE, CØMPACT
DMARK	A flag.	PURSE, CØMPACT, RELINK
E	Part of feature string.	ANALYZER, DEFINE, SEARCHF, SEARCHS, MERGE, GCHECK
ecode	Single generated feature storage.	INFLEX
F	1) Filter constant set by SAMPLE and TEST for use by GRID. 2) Part of generated feature string.	SAMPLE, TEST, GRID ANALYZER, DEFINE, SEARCHF, SEARCHS, MERGE, GCHECK
FB	Abbreviation for "function button"	SAMPLE, TEST,
fc	Feature count.	BØTH, APUT
G	Geometric relationship between strokes of a multi-stroke character.	ANALYZER, DEFINE, SEARCHD, SEARCHS, MERGE, GCHECK
GChain	Pointer into dictionary.	OPTIMIZE, ADDGRØUP, GCHECK, RELINK
GNLink	Pointer into GRØUP table.	OPTIMIZE, GCHECK, RELINK
GSLink	Pointer into GRØUP table.	OPTIMIZE, GTREE, GCHECK, RELINK
h	Heading computed between two points by DIRQ.	STRØKE
H	Upper limit for testing curvature in inflection point computation.	INFLEX
I	Indicator for primary stroke features: corner, inflection point, or loop.	STRØKE, BØTH

<u>TERM</u>	<u>MEANING</u>	<u>REFERENCE</u>
ILØC	Coordinate location on display surface for output messages.	SAMPLE
IMB	Input memory buffer for display refreshing.	GRID, SAMPLE, TEST
INKLØC	The current relative location in IMB for "ink".	GRID, SAMPLE, TEST, ANALYZER
INKØRG	The relative location that "ink" is to start in IMB.	SAMPLE, TEST, ANALYZER
k	Flag used to indicate existence of an inflection point.	INFLEX, BØTH
KB	Abbreviation for keyboard button.	SAMPLE
KCB	Abbreviation for keyboard change button.	SAMPLE
L	Lower limit for testing curvature in inflection point computation.	INFLEX
LLast	Pointer used for termination of processing.	PURGE, MERGE
LSort	Pointer into SØRT table.	ØPTIMIZE
Lthis	Pointer used for various purposes.	PURGE, MERGE, DTREE, GTREE, GCHECK
mdist	Minimum value of square of computed distance between two points.	MINPTS
NLink	The pointer to the next stroke at the save level in the list of successor strokes in the dictionary.	DEFINE, SEARCHS, PURGE, COMPACT, MERGE, DTREE, GCHECK, RELINK
Ns	Number of strokes.	GRID, ANALYZER, DEFINE

<u>TERM</u>	<u>MEANING</u>	<u>REFERENCE</u>
OV	Overflow flag.	MERGE
PChar	Output character definitions to be purged from dictionary.	PURGE
pix	Test value for index.	STRØKE
plim	Test value for index.	XØVER, MINPTS
PG	Purge flag.	MERGE
psw	RAND Tablet stylus tip switch.	GRID
Q	Feature storage location.	XØVER
qlim	Test value for index.	MINPTS
R(x)	Minimum rectangle surrounding argument; either a feature or an entire stroke.	SAMPLE, TEST, ANALYZER, INFLEX, BØTH, XØVER, SEARCHD
$\bar{R}(x)$	Center of minimum rectangle (see R(x)).	ANALYZER, BØTH, SEARCHED
Rc	Recognition count kept in dictionary for each successful match made with the associated entry.	ANALYZER, PURGE, MERGE, RELINK
S	Symbol used to denote an input stroke.	SAMPLE, TEST
S1, S2	Flags.	PURGE
savfc	Temporary storage for feature count.	XØVER
savp	Temporary storage for index p.	MINPTS
scrubflag	Flag used to indicate whether or not an input stroke is to be interpreted as an erasure.	TEST
sig	Flag.	ØPTIMIZE

<u>TERM</u>	<u>MEANING</u>	<u>REFERENCE</u>
Size(x)	Size (height and width) of rectangle surrounding argument (call R(x)).	INFLEX, SEARCHD
SLink	The pointer to the first legitimate successor stroke for multi-stroke characters in a dictionary definition.	DEFINE, SEARCHS, PURGE, CØMPACT, DTREE, GCHECK, RELINK
SLØC	The pointer to the next available successor stroke entry in the dictionary.	DEFINE, PURGE, CØMPACT, MERGE
SMflg	Smoothing flag; set in control word of display buffer in Input Memory to enable or disable smoothing.	GRID, SAMPLE, TEST
SØRTL	Entry in SØRT table composed of stroke level and associated definition.	ØPTIMIZE
SØRTR	Dictionary pointer from SØRT table for SØRTL item.	ØPTIMIZE
SW1	The program switch in GRID that is set to the appropriate function according to the value of psw and T_d , the time delay. It has the value IGN, meaning "ignore the tablet"; TB, meaning "post only the current position of the pen in IMB at TB"; and INK, meaning "filter and post the path of the pen on the tablet as long as psw is on and there is room in the buffer".	GRID
SW2	The program switch in GRID that enables or disables smoothing. It has the values of "on" and "off" according to SMFLG.	GRID

<u>TERM</u>	<u>MEANING</u>	<u>REFERENCE</u>
t or temp	A temporary storage.	STRØKE, ØPTIMIZE
tbeg	Pointer to the beginning of a stroke.	ANALYZER, STRØKE
TD	The time delay set in IMB by the calling program to specify when GRID shall give up control after a psw "off" is detected. Time is specified in units of .25 sec.	GRID
TD'	An intermediate storage for TD.	GRID
TD*	The computed value for use by GRID to effect the time delay test based upon a clock that increments in units other than .25 sec.	GRID
Thresh	Recognition count threshold. Dictionary entries with a recognition count less than the threshold are removed from the dictionary.	PURGE
tmax	Pointer to one beyond the last entry of tablet inputs in DB.	ANALYZER, STRØKE
tmore	Flag used for terminating processing in TEST mode of ANALYZER.	ANALYZER
x,y	Individual coordinates of points making up input strokes stored in DB.	GRID, STRØKE
X,Y	Individual coordinates of points making up input strokes stored in PTS table.	STRØKE, BØTH, INFLEX, XØVER
xtry	Flag used to indicate success or failure of loop or inter-section test.	XØVER, MINPTS

ADDENDUM B: EXAMPLE OF OPTIMIZE

Figure B-1 represents a sample dictionary before and after OPTIMIZE. Figures B-2 through B-22 present the contents of the dictionary and other tables at successive stages of the optimize process. Strokes are represented symbolically in the first column of the dictionary (labeled "Feature"); at each dictionary level, strokes represented by the same symbol are identical. For this sample dictionary, three dictionary entries are deleted, and nine new character definitions are added. In the table below, entries above the double line represent dictionary definitions before optimize, while entries below the double line are those added by optimize.

Table B-1. Stroke Entries for Sample Dictionary

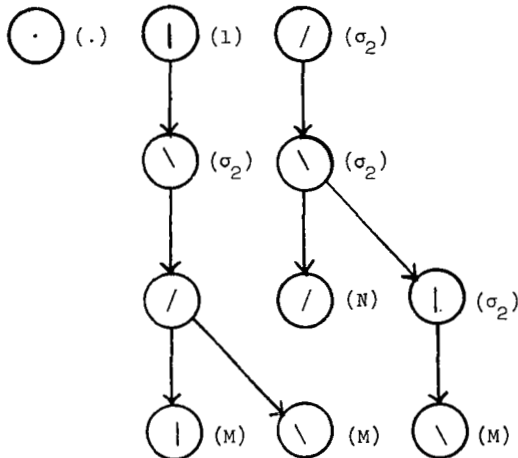
Character	Stroke 1	Stroke 2	Stroke 3	Stroke 4
.	.			
l				
N	/	\	/	
M		\	/	
M		\	/	\
M	/	\		\
l	/			
N		\		
N	/	\		
N		\	/	
M		\		\
M		\		
M	/	\		
M	/	\	/	\
M	/	\	/	

	Feature	Def	SLink	NLink
0				
1	.	.	0	0
2		1	509	0
3	/	σ_2	506	0
504	\	M	0	0
505		σ_2	504	0
506	\	σ_2	507	0
507	/	N	0	505
508	\	M	0	0
509	\	σ_2	510	0
510	/	σ_2	511	0
511		M	0	508

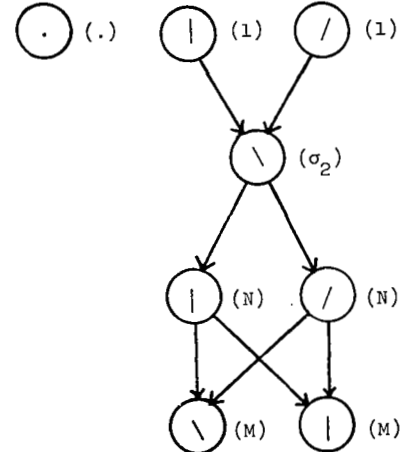
a. DICT before optimize

	Feature	Def	SLink	NLink
0				
1	.	.	0	0
2		1	509	0
3	/	1	509	0
507		N	508	510
508	\	M	0	511
509	\	σ_2	507	0
510	/	N	508	0
511		M	0	0

b. DICT after optimize



c. Tree Diagram of original Dictionary



d. Tree of Optimized Dictionary

Figure B-1. Sample Dictionary Before and After Optimization

DICT					DICTE			
	Feature	Def	SLink	NLink	DGLink	DGroup	DEQ	DMark
0								
1	.	.	0	0				
2		1	509	0				
3	/	σ_2	506	0				
504	\	M	.	0				
505		σ_2	504	0				
506	\	σ_2	507	0				
507	/	N	0	505				
508	\	M	0	0				
509	\	σ_2	510	0				
510	/	σ_2	511	0				
511		M	0	508				

SORT				
	Ch	GChain	GSLink	GNLink
0	0	0	0	0
1				
2				
3				
4				
5				

SORT		
	Sortl	Sortr
0	0	0
1	1	2
2	1	1
3	3	N
4	4	M
5	4	M
6	0	0
7		
8		

LTAB		
	LGroup	LSort
0	0	0
1	0	0
2		
3		
4		

Figure B-2. Step 1 of Optimization

The first-level section of the SORT table was made. DIREC was applied to each dictionary first stroke (strokes 1-3). LSort₁ points to SORT₀, the top bracket of the section. SORT₆ is the bottom bracket.

DICT

DICTE

	Feature	Def	SLink	NLink	DGLink	DGroup	DEQ	DMark
0								
1	.	.	0	0				
2		1	509	0	0	1		
3	/	σ_2	506	0	2	1		
504	\	M	0	0				
505		σ_2	504	0				
506	\	σ_2	507	0				
507	/	N	0	505				
508	\	M	0	0				
509	\	σ_2	510	0				
510	/	σ_2	511	0				
511		M	0	508				

GROUP

	Ch	GChain	GSLink	GNLink
0	0	0	0	0
1	1	3	0	0
2				
3				
4				
5				

SORT

	Sortl	Sortr
0	0	0
1	1	.
2	0	0
3		
4		
5		
6		
7		
8		

LTAB

	LGroup	LSort
0	0	0
1	1	0
2		
3		
4		

Figure B-3. Step 2 of Optimization

GROUP₁ was formed at level 1; its members are stroke₃ and stroke₂ (each is the first stroke of a 4-stroke "M"). The group character is '1'. LGroup₁ points to GROUP₁, the current GROUP entry at level 1. The remaining first-level SORT section has been compacted; SORT₂ is now its bottom bracket. Note that if stroke₃ had been defined as a divide sign, the group character would be " σ_1 ", and further work on this family (Figures B-4 through B-19) would have been aborted.

DICT					DICTE			
	Feature	Def	SLink	NLink	DGLink	DGroup	DEQ	DMark
0								
1	.	.	0	0				
2		1	509	0	0	1		
3	/	σ_2	506	0	2	1		
504	\	M	0	0				
505		σ_2	504	0				
506	\	σ_2	507	0				
507	/	N	0	505				
508	\	M	0	0				
509	\	σ_2	510	0				
510	/	σ_2	0	0				
511		M	0	508				

GROUP				
	Ch	GChain	GSLink	GNLink
0	0	0	0	0
1	1	3	0	0
2				
3				
4				
5				

SORT		
	Sort1	Sortr
0	0	0
1	1	.
2	0	0
3	2	N
4	3	M
5	3	M
6	0	0
7		
8		

LTAB		
	LGroup	LSort
0	0	0
1	1	0
2		2
3		
4		

Figure B-4. Step 3 of Optimization

A second-level SORT section was made. DTREE was applied to stroke_3 and stroke_2 , the members of GROUP_1 .

DICT					DICTE			
	Feature	Def	SLink	NLink	DGLink	DGroup	DEQ	DMark
0								
1	.	.	0	0				
2		1	509	0	0	1		
3	/	σ_2	506	0	2	1		
504	\	M	0	0				
505		σ_2	504	0				
506	\	σ_2	507	0	0	2		
507	/	N	0	505				
508	\	M	0	0				
509	\	σ_2	510	0	506	2		
510	/	σ_2	511	0				
511		M	0	508				

GROUP			
	Ch	GChain	GSLink
0	0	0	0
1	1	3	2
2	σ_2	509	0
3			
4			
5			

SORT		
	Sortl	Sortr
0	0	0
1	1	1
2	0	0
3	0	0
4		
5		
6		
7		
8		

LTAB	
	LGroup
0	0
1	1
2	2
3	
4	

Figure B-5. Step 4 of Optimization

GROUP₂ at level 2 was formed. Its members, stroke₅₀₉ and stroke₅₀₆, are each the second stroke of a 4-stroke "M". GROUP₂ is a successor to GROUP₁; its group character is " σ_2 ".

DICT					DICTE			
	Feature	Def	SLink	DLink	DGLink	DGroup	DEQ	DMark
0								
1	.	.	0	0				
2		1	509	0	0	1		
3	/	σ_2	506	0	2	1		
504	\	M	0	0				
505		σ_2	504	0				
506	\	σ_2	507	0	0	2		
507	/	N	0	505				
508	\	M	0	0				
509	\	σ_2	510	0	506	2		
510	/	σ_2	511	0				
511		M	0	508				

GROUP				
	Ch	GChain	GSLink	GNLink
0	0	0	0	0
1	1	3	2	0
2	σ_2	509	0	0
3				
4				
5				

SORT		
	Sortl	Sortr
0	0	0
1	1	1
2	0	0
3	0	0
4	1	N
5	2	M
6	2	M
7	0	0
8		

LTAB		
	LGroup	LSort
0	0	0
1	1	0
2	2	2
3		3
4		

Figure B-6. Step 5 of Optimization

A third-level SORT section was made. DTREE was applied to stroke₅₀₉ and stroke₅₀₆, the members of GROUP₂.

	DICT				DICTE			
	Feature	Def	SLink	NLink	DGLink	DGroup	DEQ	IMark
0								
1	.	.	0	0				
2		1	509	0	0	1		
3	/	σ_2	506	0	2	1		
504	\	M	0	0				
505		σ_2	504	0				
506	\	σ_2	507	0	0	2		
507	/	N	0	505	0	3		
508	\	M	0	0				
509	\	σ_2	510	0	506	2		
510	/	σ_2	511	0				
511		M	0	508				

GROUP				
	Ch	GChain	GSLink	GNLink
0	0	0	0	0
1	1	3	2	0
2	σ_2	509	3	0
3	N	507	0	0
4				
5				

SORT		
	Sortl	Sortr
0	0	0
1	1	1
2	0	0
3	0	0
4	2	M
5	2	M
6	0	0
7		
8		

LTAB		
	LGroup	LSort
0	0	0
1	1	0
2	2	2
3	3	3
4		

Figure B-7. Step 6 of Optimization

GROUP₃ was formed at level 3; its single member is stroke₅₀₇. GROUP₃ is a successor to GROUP₂; its group character is "N".

	DICT				DICTE			
	Feature	Def	SLink	NLink	DGLink	DGroup	DEQ	DMark
0								
1	.	.	0	0				
2		1	509	0	0	1		
3	/	σ_2	506	0	2	1		
504	\	M	0	0				
505		σ_2	504	0				
506	\	σ_2	507	0	0	2		
507	/	N	0	505	0	3		
508	\	M	0	0				
509	\	σ_2	510	0	506	2		
510	/	σ_2	511	0				
511		M	0	508				

GROUP				
	Ch	GChain	GSLink	GNLink
0				
1	1	3	2	0
2	σ_2	509	3	0
3	N	507	0	0
4				
5				

SORT		
	Sortl	Sortr
0	0	0
1	1	.
2	0	0
3	0	0
4	2	M
5	2	M
6	0	0
7	0	0
8		

LTAB		
	LGroup	LSort
0	0	0
1	1	0
2	2	2
3	3	3
4		6

Figure B-8. Step 7 of Optimization

A fourth-level SORT section was formed. DTREE was applied to stroke₅₀₇, the only member of GROUP₃. Stroke₅₀₇ has no successors in the dictionary, therefore the SORT section is empty.

DICT					DICTE			
	Feature	Def	SLink	NLink	DGLink	DGroup	DEQ	DMark
0								
1	.	.	0	0				
2		1	509	0	0	1		
3	/	σ_2	506	0	2	1		
504	\	M	0	0				
505		σ_2	504	0	0	4		
506	\	σ_2	507	0	0	2		
507	/	N	0	505	0	3		
508	\	M	0	0				
509	\	σ_2	510	0	506	2		
510	/	σ_2	511	0	505	4		
511		M	0	508				

GROUP				
	Ch	GChain	GSLink	GNLink
0	0	0	0	0
1	1	3	2	0
2	σ_2	509	4	0
3	N	507	0	0
4	σ_2	510	0	3
5				

SORT		
	Sortl	Sortr
0	0	0
1	1	.
2	0	0
3	0	0
4	0	0
5		
6		
7		
8		

LTAB	
LGroup	LSort
0	0
1	1
2	2
3	4
4	

Figure B-9. Step 8 of Optimization

GROUP₄ at level 3 was formed; its members are stroke₅₁₀ and stroke₅₀₅. GROUP₄ was added to the successor chain of GROUP₂; the chain now contains GROUP₄ and GROUP₃. The group character of GROUP₄ is " σ_2 ". Stroke₅₁₀ and stroke₅₀₅ are each third strokes of a 4-stroke "M".

DICT					DICTE			
	Feature	Def	SLink	NLink	DGLink	DGroup	DEQ	DMark
0								
1	.	.	0	0				
2		1	509	0	0	1		
3	/	σ_2	506	0	2	1		
504	\	M	0	0				
505		σ_2	504	0	0	4		
506	\	σ_2	507	0	0	2		
507	/	N	0	505	0	3		
508	\	M	0	0				
509	\	σ_2	510	0	506	2		
510	/	σ_2	511	0	505	4		
511		M	0	508				

GROUP				SORT			LTAB	
	Ch	GChain	GSLink	GNLink	Sortl		Sortr	
0	0	0	0	0	0	0	0	0
1	1	3	2	0	1	.	1	0
2	σ_2	509	4	0	0	0	0	2
3	N	507	0	0	0	0	0	3
4	σ_2	510	0	3	0	0	0	4
5					1	M	504	
					1	M	508	
					1	M	511	
					0	0	0	

Figure B-10. Step 9 of Optimization

A fourth-level SORT section was made. DTREE was applied to stroke₅₁₀ and stroke₅₀₅, the members of GROUP₄.

	DICT				DICTE			
	Feature	Def	SLink	NLink	DGLink	DGroup	DEQ	DMark
0								
1	.	.	0	0				
2		1	509	0	0	1		
3	/	σ_2	506	0	2	1		
504	\	M	0	0	0	5		
505		σ_2	504	0	0	4		
506	\	σ_2	507	0	0	2		
507	/	N	0	505	0	3		
508	\	M	0	0	504	5		
509	\	σ_2	510	0	506	2		
510	/	σ_2	511	0	505	4		
511		M	0	508	508	5		

GROUP				
	Ch	GChain	GSLink	GNLink
0	0	0	0	0
1	1	3	2	0
2	σ_2	509	4	0
3	N	507	0	0
4	σ_2	510	5	3
5	M	511	0	0

SORT		
	Sortl	Sortr
0	0	0
1	1	1
2	0	0
3		
4		
5		
6		
7		
8		

LTAB		
	LGroup	LSort
0	0	0
1	1	0
2		
3		
4		

Figure B-13. Step 12 of Optimization

The SORT sections at levels 4, 3, and 2 are found to be empty, and the level counter is stepped back to level 1. Formation of the first group tree is complete.

	DICT				DICTE			
	Feature	Def	SLink	NLink	DGLink	DGroup	DEQ	IMark
0								
1	.	.	0	0				
2		1	509	0	0	1		
3	/	σ_2	506	0	2	1		
504	\	M	0	0	0	5		
505		σ_2	504	0	0	4		
506	\	σ_2	507	0	0	2	509	
507	/	N	0	505	0	3		
508	\	M	0	0	504	5		
509	\	σ_2	510	0	506	2		
510	/	σ_2	511	0	505	4		
511		M	0	508	508	5		

	Ch	GChain	GSLink	GNLink
0	0	0	0	0
1	1	3	2	0
2	σ_2	509	4	0
3	N	507	0	0
4	σ_2	510	5	3
5	M	511	0	0

	Sortl		Sortr
0	0	0	0
1	1	.	1
2	0	0	0
3			
4			
5			
6			
7			
8			

	LGroup	LSort
0	0	0
1	1	0
2		
3		
4		

Figure B-14. Step 13 of Optimization

Procedure GCHECK was applied to GROUP₁, but performs no operation on a first-level group. GCHECK was then applied to GROUP₂; stroke₅₀₆ was marked as identical to stroke₅₀₉.

	DICT				DICTE			
	Feature	Def	SLink	NLink	DGLink	DGroup	DEQ	DMark
0								
1	.	.	0	0				
2		1	509	0	0	1		
3	/	σ_2	506	0	2	1		
504	\	M	0	0	0	5		
505		σ_2	504	0	0	4		
506	\	σ_2	507	0	0	2	509	
507	/	N	0	505	505	4	510	
508	\	M	0	0	504	5		
509	\	σ_2	510	0	506	2		
510	/	σ_2	511	0	507	4		
511		M	0	508	508	5		

	GROUP				SORT			LTAB	
	Ch	GChain	GSLink	GNLink	Sortl		Sortr	LGroup	LSort
0	0	0	0	0	0	0	0	0	0
1	1	3	2	0	1	.	1	1	0
2	σ_2	509	4	0	0	0	0		
3									
4	N	510	5	0					
5	M	511	0	0					

0	0	0
1	1	.
2	0	0
3		
4		
5		
6		
7		
8		

0	0	0
1	1	0
2		
3		
4		

Figure B-15. Step 14 of Optimization

GCHECK was applied to $GROUP_4$, the first successor to $GROUP_2$. Stroke₅₀₇ in $GROUP_3$ (the other successor to $GROUP_2$) was marked as identical to stroke₅₁₀ in $GROUP_4$. $GROUP_3$ was merged with $GROUP_4$, and entry₃ of the GROUP table is now dead (there are no pointers to it). The group character of $GROUP_4$ was changed from " σ_2 " to "N" because $GROUP_3$'s character was "N".

DICT					DICTE			
	Feature	Def	SLink	NLink	DGLink	DGroup	DEQ	DMark
0								
1	.	.	0	0				
2	!	1	509	0	0	1		
3	/	σ_2	506	0	2	1		
504	\	M	0	0	0	5	508	
505		σ_2	504	0	0	4		
506	\	σ_2	507	0	0	2	509	
507	/	N	0	505	505	4	510	
508	\	M	0	0	504	5		
509	\	σ_2	510	0	506	2		
510	/	σ_2	511	0	507	4		
511		M	0	508	508	5		

GROUP				
	Ch	GChain	GSLink	GNLink
0	0	0	0	0
1	1	3	2	0
2	σ_2	509	4	0
3				
4	N	510	5	0
5	M	511	0	0

SORT		
	Sortl	Sortr
0	0	0
1	1	1
2	0	0
3		
4		
5		
6		
7		
8		

LTAB		
	LGroup	LSort
0	0	0
1	1	0
2		
3		
4		

Figure B-16. Step 15 of Optimization

GCHECK was applied to GROUP₅. Stroke₅₀₄ was marked as identical to stroke₅₀₈. GCHECK has now been applied to all groups in the tree.

	DICT				DICTE			
	Feature	Def	SLink	NLink	DGLink	DGroup	DEQ	DMark
0								
1	.	.	0	0				
2		1	509	0	0	1		
3	/	1	509	0	2	1		
504	\	M	0	0	0	5	508	
505		σ_2	504	0	0	4		
506	\	σ_2	507	0	0	2	509	1
507	/	N	0	505	505	4	510	
508	\	M	0	0	504	5		
509	\	σ_2	510	0	506	2		
510	/	σ_2	511	0	507	4		
511		M	0	508	508	5		

	GROUP				SORT			LTAB	
	Ch	GChain	GSLink	GNLink	Sortl		Sortr	LGroup	LSort
0	0	0	0	0	0	0	0	0	0
1	1	3	2	0	1	.	1	1	0
2	σ_2	509	4	0	0	0	0		
3									
4	N	510	5	0					
5	M	511	0	0					

Figure B-17. Step 16 of Optimization

Procedure RELINK was applied to GROUP₁. Stroke₃ in this group was redefined from " σ_2 " to the group character "1". The dictionary successor chain to members of GROUP₁ contains only stroke₅₀₉; stroke₅₀₆, which is identical to stroke₅₀₉, was not included in the successor chain, but was marked for deletion. The SLinks of both members of GROUP₁ point to the common successor chain.

DICT					DICTE			
	Feature	Def	SLink	NLink	DGLink	DGroup	DEQ	DMark
0								
1	.	.	0	0				
2		1	509	0	0	1		
3	/	1	509	0	2	1		
504	\	M	0	0	0	5	508	
505		σ_2	504	510	0	4		
506	\	σ_2	505	0	0	2	509	1
507	/	N	0	505	505	4	510	1
508	\	M	0	0	504	5		
509	\	σ_2	505	0	506	2		
510	/	σ_2	511	0	507	4		
511		M	0	508	508	5		

GROUP				
	Ch	GChain	GSLink	GNLink
0	0	0	0	0
1	1	3	2	0
2	σ_2	509	4	0
3				
4	N	510	5	0
5	M	511	0	0

SORT		
	Sortl	Sortr
0	0	0
1	1	.
2	0	0
3		
4		
5		
6		
7		
8		

LTAB		
	LGroup	LSort
0	0	0
1	1	0
2		
3		
4		

Figure B-18. Step 17 of Optimization

RELINK was applied to GROUP₂. The dictionary successor chain to GROUP₂ members contains stroke₅₀₅ and stroke₅₁₀. Stroke₅₀₇, which is identical to stroke₅₁₀, is marked for deletion.

	DICT	DICTE						
	Feature	Def	SLink	NLink	DGLink	DGroup	DEQ	DMark
0								
1	.	.	0	0				
2		1	509	0	0	1		
3	/	1	509	0	2	1		
504	\	M	0	0	0	5	508	1
505		N	508	510	0	4		
506	\	σ_2	505	0	0	2	509	1
507	/	N	508	505	505	4	510	1
508	\	M	0	511	504	5		
509	\	σ_2	505	0	506	2		
510	/	N	508	0	507	4		
511		M	0	0	508	5		

	GROUP			
	Ch	GChain	GSLink	GNLink
0	0	0	0	0
1	1	3	2	0
2	σ_2	509	4	0
3				
4	N	510	5	0
5	M	511	0	0

	SORT	
	Sortl	Sortr
0	0	0
1	1	.
2	0	0
3		
4		
5		
6		
7		
8		

	LTAB	
	LGroup	LSort
0	0	0
1	1	0
2		
3		
4		

Figure B-19. Step 18 of Optimization

RELINK was applied to GROUP₄. Both members of GROUP₄ (stroke₅₁₀ and stroke₅₀₅) were redefined from " σ_2 " to the group character "N". The dictionary successor chain which is common to both stroke₅₁₀ and stroke₅₀₅ contains stroke₅₀₈ and stroke₅₁₁. Stroke₅₀₄, identical to stroke₅₀₈, was marked for deletion.

RELINK was then applied to GROUP₅. The successor chain for its members is empty. All groups have been relinked; processing of this dictionary family is completed.

	DICT				DICTE			
	Feature	Def	SLink	NLink	DGLink	DGroup	DEQ	DMark
0								
1	.	.	0	0	0	1		
2		1	509	0	0	1		
3	/	1	509	0	2	1		
504	\	M	0	0	0	5	508	1
505		N	508	510	0	4		
506	\	σ_2	505	0	0	2	509	1
507	/	N	508	505	505	4	510	1
508	\	M	0	511	504	5		
509	\	σ_2	505	0	506	2		
510	/	N	508	0	507	4		
511		M	0	0	508	5		

DICT				SORT			LTAB	
	Ch	GChain	GSLink	GNLink	Sortl	Sortr	LGroup	LSort
0	0	0	0	0	0	0	0	0
1	.	1	0	0	0	0	1	0
2								
3								
4								
5								
6								
7								
8								

Figure B-20. Step 19 of Optimization

Processing of the next dictionary family has begun. A new $GROUP_1$ was formed from the first-level SORT section. $Stroke_1$ is the only member of this group.

DICT					DICTE			
	Feature	Def	SLink	NLink	DGLink	DGroup	DEQ	DMark
0								
1	.	.	0	0	0	1		
2		1	509	0	0	1		
3	/	1	509	0	2	1		
504	\	M	0	0	0	5	508	1
505		N	508	510	0	4		
506	\	σ_2	505	0	0	2	509	1
507	/	N	508	505	505	4	510	1
508	\	M	0	511	504	5		
509	\	σ_2	505	0	506	2		
510	/	N	508	0	507	4		
511		M	0	0	508	5		

GROUP				
	Ch	GChain	GSLink	GNLink
0	0	0	0	0
1	.	1	0	0
2				
3				
4				
5				

SORT		
	Sortl	Sortr
0	0	0
1	0	0
2	0	0
3		
4		
5		
6		
7		
8		

LTAB		
	LGroup	LSort
0	0	0
1	1	0
2		1
3		
4		

Figure B-21. Step 20 of Optimization

A second-level SORT section was made by applying DTREE to stroke₁, the only member of GROUP₁. The SORT section is empty, and the new group tree is complete. Because GROUP₁ has no successor groups, GTREE and RELINK are not operated, and processing of the second family is complete. The first-level SORT section is also empty, so there are no more dictionary families to process.

	DICT				DICTE			
	Feature	Def	SLink	NLink	DGLink	DGroup	DEQ	DMark
0								
1	.	.	0	0				
2		1	509	0				
3	/	1	509	0				
504								
505								
506								
507		N	508	510				
508	\	M	0	511				
509	\	σ_2	507	0				
510	/	N	508	0				
511		M	0	0				

GROUP				SORT			LTAB	
	Ch	GChain	GSLink	GNLink	Sortl	Sortr	LGroup	LSort
0								
1								
2								
3								
4								
5								
6								
7								
8								

Figure B-22. Final Result of Optimization

Procedure COMPACT has been operated. Stroke₅₀₄, stroke₅₀₆ and stroke₅₀₇ were deleted. Stroke₅₀₅ was moved down and became stroke₅₀₇. The SLink of stroke₅₀₉, which pointed to stroke₅₀₅, now points to stroke₅₀₇.

(last page)

ADDENDUM C: NEW TECHNOLOGY

It is difficult to say what, in particular, about this program lies within the realm of new technology. Rather than specific algorithms or solutions to particular problems, it is the general approach that is unique--the combination of existing methodology that is new.

At the actual working level of the program, two things in particular are different from earlier approaches to character recognition. One is the feature-extraction technique, including corner-detection; the other is the use of the dictionary to provide separation between adjacent characters, instead of some other measure.

NOV 13 1971
AERONAUTICS AND SPACE ADMINISTRATION
WASHINGTON, D.C. 20546



POSTMASTER: If Undeliverable (See
Postal Manual) Do Not

"The aeronautical and space activities of the United States shall be conducted so as to contribute . . . to the expansion of human knowledge of phenomena in the atmosphere and space. The Administration shall provide for the widest practicable and appropriate dissemination of information concerning its activities and the results thereof."

— NATIONAL AERONAUTICS AND SPACE ACT OF 1958

NASA SCIENTIFIC AND TECHNICAL PUBLICATIONS

TECHNICAL REPORTS: Scientific and technical information considered important, complete, and a lasting contribution to existing knowledge.

TECHNICAL NOTES: Information less broad in scope but nevertheless of importance as a contribution to existing knowledge.

TECHNICAL MEMORANDUMS: Information receiving limited distribution because of preliminary data, security classification, or other reasons.

CONTRACTOR REPORTS: Scientific and technical information generated under a NASA contract or grant and considered an important contribution to existing knowledge.

TECHNICAL TRANSLATIONS: Information published in a foreign language considered to merit NASA distribution in English.

SPECIAL PUBLICATIONS: Information derived from or of value to NASA activities. Publications include conference proceedings, monographs, data compilations, handbooks, sourcebooks, and special bibliographies.

TECHNOLOGY UTILIZATION PUBLICATIONS: Information on technology used by NASA that may be of particular interest in commercial and other non-aerospace applications. Publications include Tech Briefs, Technology Utilization Reports and Notes, and Technology Surveys.

Details on the availability of these publications may be obtained from:

SCIENTIFIC AND TECHNICAL INFORMATION DIVISION
NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
Washington, D.C. 20546